UNDERSTANDING HOW DEVELOPERS EDIT SOURCE CODE

A dissertation submitted to

Kent State University in partial

fulfillment of the requirements for the

Degree of Doctor of Philosophy

by

Joshua Allan Charles Behler

August 2026

Dissertation written by

Joshua A.C. Behler

B.S., Kent State University, 2021

M.S., Kent State University, 2023

Ph.D., Kent State University, 2026

Approved by

_____, Chair, Doctoral Dissertation Committee
Jonathan I. Maletic

_____, Members, Doctoral Dissertation Committee
Qiang Guan

_____
Kwangtaek Kim

_____
Bonita Sharif

_____
Jocelyn Folk

_____
Susan Fisk

Accepted by

_____, Chair, Department of Computer Science
Mikhail Nesterenko

_____, Dean, College of Arts and Sciences
Mandy Munro-Stasiuk

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

There are many people to whom I owe thanks and appreciation for helping me accomplish this effort.

First and foremost, I would like to thank my wife, Rebecca (Turner) Behler, for her love and support. I would not have been able to accomplish nearly as much as I have without you. You give my life great meaning.

I would like to thank my parents Brian and Elizabeth Behler, and the rest of my family, for their support and encouraging me to pursue my goals.

I would like to thank my advisor, Prof. Jonathan Maletic, for introducing me to academia, mentoring me, helping me grow as a researcher and an educator, and granting me wonderful opportunities to see the world. I could not have asked for a better advisor.

I would also like to thank all of the other mentors who have advised me and worked with me throughout my education (in no particular order): Dr. Michael Collard, Dr. Bonita Sharif, Dr. Michael Decker, and Dr. Drew Guarnera. Your knowledge has been invaluable.

I would like to thank my fellow graduate students in the SDML lab here at Kent State (in no particular order), Ali Al-Ramadan, John Sipahioglu, Kyle Rossi, Giovanni Villalobos, and Syreen Banabilah for their help and friendship as we all navigate academia.

Lastly, I would like to thank the members of my defense committee, Dr. Qiang Guan, Dr. Kwangtaek Kim, Dr. Bonita Sharif, Dr. Jocelyn Folk, and Dr. Susan Fisk for taking time out of their busy schedules to read this dissertation and hear my defense.

Joshua Allan Charles Behler

July 2, 2026, Kent, Ohio

CHAPTER 1

Introduction

Software engineering research has produced a large body of work exploring how code evolves and is maintained over time. However, this work largely focuses on coarse-level analysis of code commits, revisions, and pull requests. Examining code at this scale analyzes what code becomes but not how code is written. Software developers do not sit down at their computers and instantly create a new commit. There is a complex problem-solving process – with developers sitting at desks typing on keyboards while solving programming problems – that is poorly understood. The moment-to-moment interactions between developers and their development environments (i.e., key presses, mouse cursor movements, and incremental edits) are absent from mental models of software development behavior [Pennington 1987a; Pennington 1987b; Von Mayrhauser and Vans 1995; Soloway and Ehrlich 1984]. To better understand how software is actually written, it is necessary to study developer behavior at a finer level of granularity.

Studying developer behavior while editing software requires the ability to capture both observable actions and the underlying cognitive processes driving them. Traditional data sources, such as version control systems and commits, provide indirect snapshots that represent coarse-grained information about developer activity, making them insufficient for eye-tracking purposes.

The main objective of this work is to explore, define categories for, and collect the moment-to-moment edits made to source code for use in further software evolution and program comprehension research.

1.1 Motivation

Eye-tracking has emerged as a powerful method for studying developer behavior and understanding the mental models of software development [Bednarik and Tukiainen 2006; Uwano, Nakamura, Monden, and Matsumoto 2006; Yusuf, Kagdi, and Maletic 2007; Sharif and Maletic 2010; Sharafi, Soh, and Guéhéneuc 2015; Obaidellah, Al Haek, and Cheng 2018; Grabinger, Hauser, Wolff, and Mottok 2024; Grabinger et al. 2025]. By capturing eye movements and visual focus, researchers can infer comprehension processes and cognitive effort in real time.

Eye-tracking in software engineering has evolved from early studies using small, static code snippets to more recent work supporting dynamic and interactive development environments. Eye-tracking allows researchers to understand the most time-consuming part of software development – reading and understanding code [Minelli, Mocci, and Lanza 2015; Xia, Bao, Lo, Xing, Hassan, and Li 2018].

Despite these advancements, there remains a large gap in current eye-tracking for software engineering research – tracking a developer while they write and develop code. Although reading and understanding code take up most of a developer's time [Minelli, Mocci, and Lanza 2015; Xia, Bao, Lo, Xing, Hassan, and Li 2018], editing is central to software development tasks such as feature implementation, debugging, and refactoring.

Understanding how developers engage with code while editing is essential to forming a complete model of programming behavior.

The primary issue in eye-tracking while editing code is mapping gaze data to a continuously changing stimulus. As mentioned previously, eye-tracking in software engineering is primarily performed on static images of simple software. The EMIP dataset is a well-known example of eye-tracking data collected on a still image of source code [Bednarik et al. 2020]. Still images of code are sufficient for certain objectives, but do not match what real developers see every day in their jobs. The issue with using a more dynamic stimulus is the difficulty of mapping gaze data to the correct region of interest within the code. The smallest shift in the document throws off every region of interest, meaning a whole new set of regions must be calculated, since there is no way to pre-determine every region's position. Current methods, such as the iTrace Infrastructure [Sharif and Maletic 2016; Guarnera, Bryant, Mishra, Maletic, and Sharif 2018], use bounding-box computation to dynamically obtain bounding boxes for the moving code stimulus. Instead of relying on a still image of code, the code can be loaded into an editing software. Using that software's API, the x and y pixel coordinates from the gaze data can be mapped to line and column information in real time. Then that line and column information can be mapped to a specific token after the eye-tracking session is completed. This innovation enables eye-tracking for software engineering studies to be performed within a full, traditional development environment with the ability to scroll through large files, switch files on demand, use the mouse to highlight important text, and supports

almost all other features within the editor. More information on the iTrace Infrastructure is detailed in CHAPTER 3.

However, iTrace has one key limitation with this approach. Because mapping the line and column to the specific token in the source code cannot be done in real time, the code must remain consistent throughout the entire development process – otherwise, the resulting data will be skewed. Unlike current read-only scenarios, editing introduces structural changes to the stimulus as tokens are inserted, modified, or deleted. This means that editing the stimulus while using iTrace is currently not allowed, and a new enhancement to eye-tracking infrastructure is necessary to support the study of developers while they edit software.

A simple approach is to capture a snapshot of the code after every single change to the text. However, even a short five-minute tracking session produces a huge volume of data that does not preserve the contextual relationships between edits. Figure 1.1 showcases a simple example of a developer editing the type in a variable declaration. In the example, after the very first text change, the syntactic meaning of the line changes from declaring a new integer variable `x` with an initial value of 0 to an already existing variable `intx` being set to 0. Looking at all the text changes, it is clear that the developer wants to change the type of variable `x`, but without the context of the total set of text changes, any assumption made purely from the syntax change between any two consecutive text changes is flawed compared to the final intention of the developer.

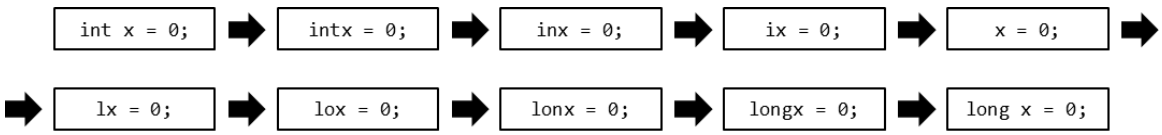


Figure 1.1. An example edit of source code that shows a developer changing the variable `x` from an `int` to a `long`. In this example, the developer placed the cursor to the left of the `x` and started their editing

Identifying every textual change to source code does not provide sufficient syntactic or semantic context for any meaningful analysis of developer intent. Conversely, as discussed earlier, looking at changes at the commit level is too coarse and does not preserve all developer intent during editing. Thus, researchers need a representation of code edits that captures changes at the most atomic level while still preserving semantic intent.

1.2 Research Questions

This work addresses the following research questions:

RQ 1. How do developers write and edit source code?

- RQ 1.1. How do developers create changes to source code?
- RQ 1.2. How does writing source code compare to writing prose?

RQ 2. What constitutes an atomic edit of source code?

- RQ 2.1. What do developers consider an atomic edit?
- RQ 2.2. Can atomic edits be programmatically identified?

RQ 3. How can eye-tracking studies map gaze data to a dynamically edited stimulus?

1.3 Research Contributions

The contributions of this work are as follows.

- The formal definition of different levels of change in software development – text events, code edits, code changes – for use in further software evolution and program comprehension research
- Custom software to conduct surveys with code and prose editing tasks for use in this work and future studies on developer editing behavior
- The creation of a large data set from 235 developers, containing over 120 hours of program editing information and over 3 hours of prose writing information
- Empirical analysis into the behavior of developers and how they interact with their keyboard/mouse while editing, marking a first step towards understanding the mental model of source code editing
- Custom software to replay responses from the survey and manually segment changes into distinct source code edits
- A reference set of edits created by expert developers from sample survey responses, used to explore the nature of source code edits
- The creation and evaluation of several algorithms to automatically create distinct source code edits from continuous editing data
- Enhancements to the iTrace Infrastructure, allowing for conducting eye-tracking studies involving an editing component, along with a pilot study that tests these enhancements

1.4 Organization of the Dissertation

This dissertation is organized as follows: CHAPTER 2 presents related work that explores how code changes and evolves. CHAPTER 3 details an overview of the iTrace Infrastructure and how editing support will fit into it. CHAPTER 4 describes how this work defines a source code edit and its constituents. CHAPTER 5 details the survey conducted by this work to gather data on how developers edit source code. CHAPTER 6 identifies developer editing behavior as collected and analyzed from survey responses. CHAPTER 7 explores how expert developers define source code edits and how edits can be programmatically identified. CHAPTER 8 details efforts to add support to the iTrace Infrastructure for conducting eye-tracking studies that allow participants to edit the stimulus, alongside a pilot eye-tracking study to demonstrate the efficacy of these additions. CHAPTER 10 presents planned future work on understanding developer behavior and supporting editing during eye-tracking studies.

CHAPTER 2

Background and Related Work

This work is an extension of various software engineering research topics; all centered on software evolution and program comprehension. The research discussed in this chapter is organized around these themes, as they all play a role in evaluating how developers edit software. First, prior work on code changes and semantic change detection is presented, including studies that classify and characterize source code modifications. Then, techniques for detecting differences between code versions are discussed. Prior eye-tracking studies of program comprehension, which motivate the need to study developers while they work on code, are then explored. Lastly, foundational work on mental models in software engineering is examined. Together, these lines of work establish the foundation and motivation for this dissertation.

2.1 Code Changes

The nature of code changes and software has been a heavily studied field. In his foundational works, Lehman defined a set of "Laws of Software Evolution," the first of which concerns software change: "A program that is used…undergoes continual change or becomes progressively less useful" [Lehman 1980; Lehman 1996]. Subsequent research has sought to examine Lehman's observations by investigating how best to identify and

categorize software changes. Later, Kagdi et al. developed techniques for analyzing software systems using version control systems [Kagdi, Collard, and Maletic 2007]. This led to a large body of work on mining software repositories, in which code evolution is studied through commits and revisions. Studies that investigate commit-level observations are discussed in the next subsection. However, repository-based representations of change provide only a partial view of the development process and do not capture the continuous sequence of edits that lead to the code as it currently stands. Robbes and Lanza attempt to address this limitation by proposing a methodology to capture changes to the source code as they are made [Robbes and Lanza 2007]. Their tool, SpyWare, captures snapshots of the source code when specific change types are made, such as the addition or removal of a method or variable. This approach is closely related to this work but still focuses on a too coarse level of code change (i.e., statement-level vs. token-level). In later work, Chen et al. explore what they call micro-changes [Chen, Lanza, and Hayashi 2024]. Micro-changes are small code-change operations that can be extracted from larger code diffs and represent a smaller unit of software change. In their work, they explore only a preliminary set of micro-changes that describe how conditional statements evolve, but note that micro-changes can be defined across all aspects of source code. Examples of the micro-changes they define are *Add Conditional Statement*, *Extend If with Else*, and *Reverse Condition*.

Developers' responses to code change have also been studied. Tao et al. evaluate Microsoft employees' ability to understand the content and purpose of code changes, asking participants questions such as "What is the rationale behind this code change?" and

"Does this change introduce poor design, or break the current design?" [Tao, Dang, Xie, Zhang, and Kim 2012].

2.2 Detecting Semantic Features in Changes

One of the most common types of research on changes to source code is detecting semantic meaning or features within code changes. For example, refactorings made to software systems are commonly identified through the commits or merges within source code repositories. Many different tools exist to identify such refactorings. One of the first tools developed to automatically detect refactorings is RefactoringCrawler, a plugin for the Eclipse IDE that detects refactorings within a software component [Dig, Comertoglu, Marinov, and Johnson 2006]. Other, more recent tools also exist. RefDiff is a tool that can detect 13 "well-known" refactoring types, including extract method, extract superclass, and rename method within Java systems [Silva and Valente 2017]. RefDiff works across different revisions of a git repository by calculating a code-similarity threshold between versions and applying it to specific heuristics for each potential refactoring. Another tool, RefactoringMiner, is arguably the most prolific tool for identifying refactorings. RefactoringMiner has been widely used for over a decade and is notable for detecting 40 distinct refactorings across all levels of code, without relying on code-similarity metrics [Tsantalis, Guana, Stroulia, and Hindle 2013; Tsantalis, Mansouri, Eshkevari, Mazinanian, and Dig 2018; Tsantalis, Ketkar, and Dig 2022].

Refactoring is not the only feature that is identified through code evolution. Levin and Yehudai classify changes to software systems in relation to the testing suite, identifying whether a bugfix within the source code has a corresponding update to the testing suite.

They find that, more often than not, there is no corresponding change to the test suite [Levin and Yehudai 2017]. Hindle et al. explore the dynamic between large and small code commits and identify that larger commits appear to be perfective in nature, while smaller commits appear to be corrective [Hindle, German, and Holt 2008]. Dig and Johnson explore how APIs evolve by categorizing changes across various APIs, finding that most breaking API updates are due to refactorings that affect the public-facing end of the API [Dig and Johnson 2006]. While this work uses a manual analysis process rather than relying on automated code analysis, the system implementation was analyzed alongside artifacts such as documentation and release notes. Their results also indicate that the detection of refactorings is a good indicator of potential breaking API changes, which future studies investigated more closely [Brito, Xavier, Hora, and Valente 2018; Kula, Ouni, German, and Inoue 2018]. Pan et al. define 27 bug-fix patterns that can be extracted from source code by examining the syntactic and semantic meaning of code related to bug fixes [Pan, Kim, and Whitehead 2009]. These patterns are organized by nine specific syntactic features and how said features are added/changed over time within the source code.

2.3 Detecting Changes in Source Code

A common tool for analyzing different versions of source code is file differencing. Developers have used differencing tools for years to highlight differences between files and help understand what has changed between versions. A simple form of differencing is textual differencing, in which the text between two versions of a file is compared at the line or word level. Textual differencing, while fast and efficient [Myers 1986], often reports entire lines being changed for finer-grained changes. Syntactic differencing, a

programming syntax-aware differencing approach, provides more specific differencing information for source code files. Syntactic differencing operates on an abstract syntax tree (AST) representation of the source code rather than on the text itself. Syntactic differencing is significant to software engineering research, as it can provide specific information about what has changed between versions of source code.

Many different implementations of syntactic differencing exist, but most rely on calculating the shortest possible edit script. An edit script is a set of edits to nodes within an AST that transforms it into a new tree, meaning the shortest edit script is the smallest set of edits that can be made on one tree to turn it into a target tree. ChangeDistiller, an early tool for Java syntactic differencing, uses a simple AST in which statements are the leaf nodes [Fluri, Wursch, PInzger, and Gall 2007]. This approach, while specific to software development, is limited to reporting differences at the statement level. A later tool, GumTree, utilizes heuristics to achieve shorter edit scripts, provides human-readable output formats, and supports many programming languages [Falleri, Morandat, Blanc, Martinez, and Monperrus 2014; Martinez, Falleri, and Monperrus 2023; Falleri and Martinez 2024]. srcDiff, the tool used in this work to analyze source code edits, uses a set of differencing rules to identify the optimal difference [Decker, Collard, Volkert, and Maletic 2020]. srcDiff is notable for its XML-based srcML output format [Collard et al. 2011; Collard et al. 2013]. The XML output from srcDiff contains not just the differencing information, but also the original text of both versions of the source code embedded within the XML, including all original whitespace and comments. This makes srcDiff ideal for analyzing developer edits, as they often add and modify whitespace and comments.

2.4 Prior Eye-Tracking Work

Due to eye-tracking's emergence as an effective means of studying program comprehension, there is a large body of work exploring how developers read and understand source code. Some examples are listed below.

Binkley et al. utilize eye-tracking in multiple parts of a larger investigation into which identifier styling convention is better for comprehension [Binkley, Davis, Lawrie, Maletic, Morrell, and Sharif 2013]. Of particular interest is the "Eye Tracker Code Study," where participants were asked to read C++ code containing a mixed set of styled identifiers, and then later recall what identifiers they could remember.

Turner et al. use eye-tracking to compare comprehension of C++ source code with that of Python source code [Turner, Falcone, Sharif, and Lazar 2014]. They present participants with simple code snippets and ask them to summarize and identify bugs in the code. Their results show that participants had higher accuracy when solving C++ tasks but needed more fixations overall to solve them compared to Python.

Segedinac et al. also examine Python code using eye-tracking, evaluating comprehension of ten different statements within the language, along with other features such as the average number of calls per snippet and the average length of comments [Segedinac, Savić, Zeljković, Slivka, and Konjović 2024]. These features and the corresponding comprehension results are then used to train a machine learning model to predict a Python file's comprehensibility based on its contents.

Aljehane et al. use eye-tracking to investigate if there is a difference in how novice and expert developers read source code [Aljehane, Sharif, and Maletic 2021]. Analyzing a

previously collected eye-tracking dataset, they identify how much focus the two groups give to specific elements within the source code, finding a correlation between skill and which elements of the source code developers spend time reading.

In all the above studies, the authors' eye-tracking data is on read-only code. However, most – if not all – of the above studies could benefit from a study that allows the developer to edit/write source code.

Several tools help collect editing data for eye-tracking studies. gazel [Fakhoury, Roy, Pines, Cleveland, Peterson, Arnaoudova, Sharif, and Maletic 2021] and CodeGRITS [Tang, An, Chen, Bansal, Huang, McMillan, and Li 2024] are two IDE plugins that collect edit information during an eye-tracking session. However, both plugins use a simple edit-collection approach, capturing an entire code snapshot every time a change is made to the text. Additionally, gazel is a plugin for the now-discontinued Atom text editor and is no longer receiving support.

The abundance of potential for editing analysis in eye-tracking studies and the lack of sophisticated, well-developed tools for collecting editing information highlight the need for a well-defined, research-based approach to enable eye-tracking of source code editing.

2.5 Mental Models of Program Comprehension

In software engineering, mental models are internal representations that developers create in their minds to understand code, programming tasks, and the design of systems. Developers must form a mental model of a software system to use and/or modify it, with the complexity of the model depending on the complexity of the scenario in which the developer interacts with the system.

Numerous mental models have been developed in previous work, each formulated for a specific task or job a developer might undertake. Letovsky's model is an opportunistic model that focuses on how developers understand unfamiliar systems [Letovsky 1987]. It posits that developers do not follow a linear strategy when reading code. Instead, their attention moves around the software depending on what is most immediately useful to the developer based on hypotheses and goals. Eye-tracking work by Busjahn et al. observes reading patterns consistent with opportunistic models of how developers read code, with expert developers reading less linearly than novices [Busjahn, Bednarik, Begel, Crosby, Paterson, Schulte, Sharif, and Tamm 2015].

In his work, Brooks proposes a top-down view of software understanding, where developers rely on broader, prior knowledge of a system and domain concepts, and then work their way down to smaller, specific parts in the system [Brooks 1983]. Soloway and Ehrlich apply this approach to code, empirically investigating how developers understand parts of the code itself [Soloway and Ehrlich 1984]. Their work focuses on code constructs such as loops, conditionals, and code structures, and how developers use them to understand the code they are working with and utilizes an empirical study with human subjects to corroborate their approach. A model by Pennington splits software comprehension into two approaches: a bottom-up program model of the source code, where lower-level code and execution details build to structural understanding, and a situation model that focuses on the semantic purpose of the program [Pennington 1987a]. Von Mayrhauser and Vans propose a three-model approach, consisting of an opportunistic top-down model, an opportunistic bottom-up model, and a systematic bottom-up model, where

expiring comprehension comes from the coordination of multiple cognitive processes and does not rely on a single strategy [Von Mayrhauser and Vans 1995; von Mayrhauser and Vans 1997].

Mental models apply differently to different tasks, and the mental strategies a developer uses depend on what they are trying to accomplish. However, these models are based on comprehension of relatively stable code snapshots. Current mental models rely on the source code being a static element that does not change rapidly, and do not model the dynamics of editing or rapid code evolution that are introduced during active programming tasks. Artifacts such as run-time exceptions, unintended behavior, and compilation errors introduce new cognitive efforts that have been unexplored in mental model understandings of software development.

CHAPTER 3

The iTrace Infrastructure

iTrace[1] is an eye-tracking infrastructure designed to conduct eye-tracking studies for software engineering research. iTrace's main contribution to eye-tracking research is the ability to conduct studies on dynamic source code stimulus directly within an IDE [Walters, Falcone, Shibble, and Sharif 2013; Shaffer, Wise, Walters, Müller, Falcone, and Sharif 2015; Sharif and Maletic 2016]. This chapter details the various tools and plugins that make up the iTrace Infrastructure. Figure 3.1 provides a visual overview of the entire iTrace Infrastructure.

[1] See https://www.i-trace.org/

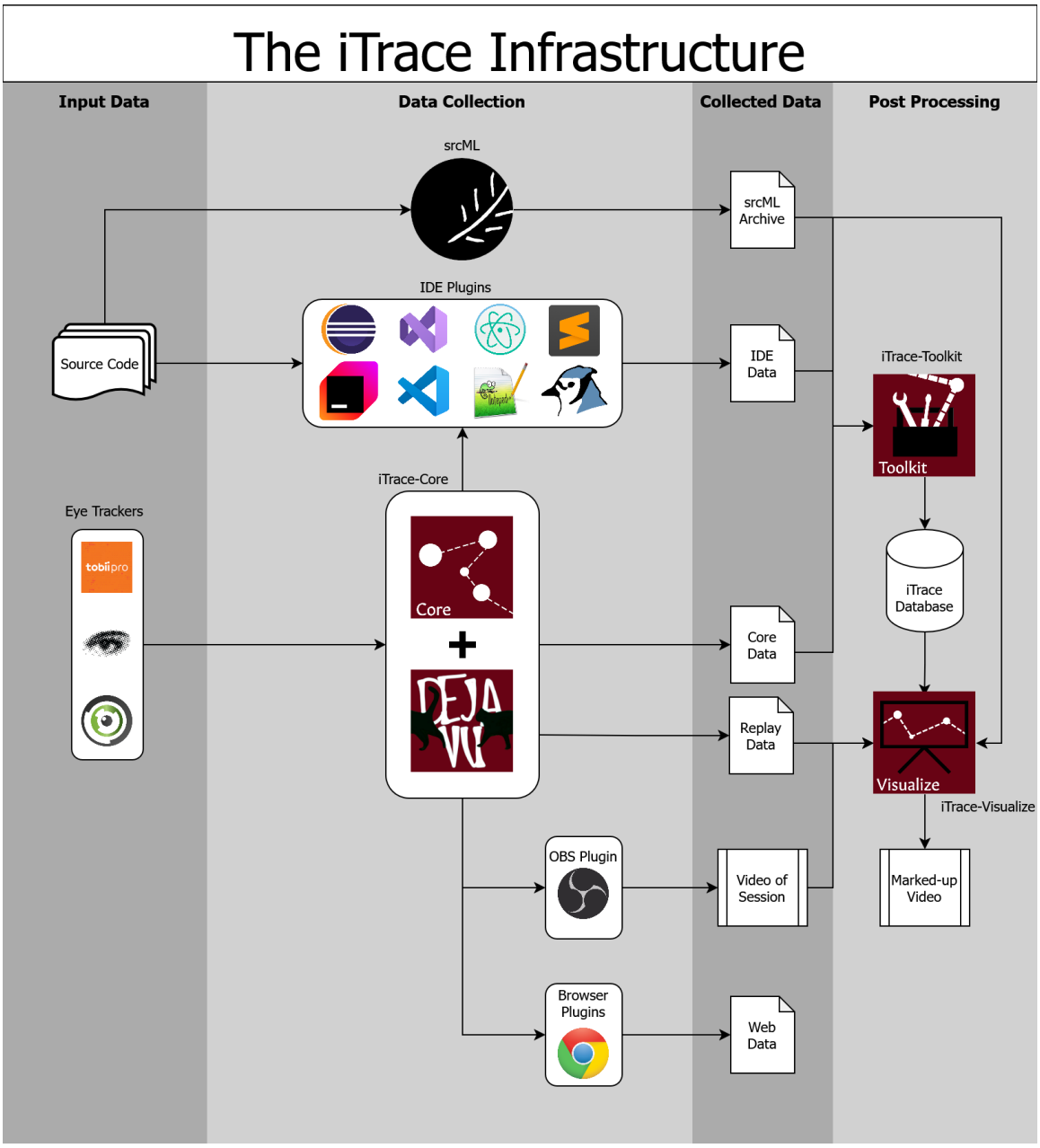


Figure 3.1. A diagram of the iTrace Infrastructure. All supported software and hardware, tools, and data files are shown

3.1 Data Collection

iTrace assists with collecting eye-tracking data during an eye-tracking session. iTrace supports eye-trackers from Tobii, SmartEye, and GazePoint. Current work is also being done to support webcam eye-tracking in iTrace [Kozak 2026].

3.1.1 iTrace-Core

iTrace-Core [Guarnera, Bryant, Mishra, Maletic, and Sharif 2018] is the central tool within the iTrace Infrastructure. iTrace-Core's main responsibility is to serve as the central server for all tools during the eye-tracking session. Before eye-tracking, iTrace-Core allows a researcher to set up the experiment information, including the participant's ID, the ID of the task about to be performed, and the name of the current researcher. The researcher can also do an iTrace calibration test, which will collect accuracy information on the current eye-tracker's performance. This calibration is separate from any eye-tracker-specific calibration and is not used to calibrate the eye-tracker's output before a recording session. During eye-tracking, iTrace-Core receives data from the eye-tracker and saves it to the *core data file* in XML format. This file contains the following data:

- Information on the current session and task
- The eye-tracking screen's width and height
- Information about the eye-tracker's make, model, and specifications
- A list of eye movement snapshots (gazes)
    - Timestamps of when the gaze occurred according to both the eye-tracker and when iTrace-Core received the gaze data
    - The x and y pixel coordinate on the screen of the gaze

- The x, y, and z coordinates of the participant's pupils in real space
- The diameter of the participant's pupils

During eye-tracking, iTrace-Core also relays information to other data-collection tools.

3.1.2 iTrace IDE Plugins

iTrace enables researchers to execute eye-tracking studies on code within real development environments. A key issue with using such a dynamic stimulus is the inability to map eye-gaze data to specific tokens in the source code. If the participant even scrolls down a single line in the editor, the pixel bounding boxes of everything in the source code change and become obsolete.

To conduct such studies, the gaze data needs to be mapped to the IDE stimulus in real-time. To accomplish this, iTrace maintains several plugins for various popular code editing environments. Currently, iTrace supports the following IDEs: Eclipse, Visual Studio, the JetBrains family of editors, Visual Studio Code, Notepad++, Sublime, and Atom (now deprecated). During an eye-tracking session, iTrace-Core sends every gaze event it receives from the eye-tracker to the IDE plugin. The IDE plugin then maps the x and y pixel coordinates from the gaze data to a specific line and column character position within the currently open source code file. This data is then written into a *plugin data file*. The plugin file contains the following data:

- The name of the coding environment that contained the code stimulus
- A timestamp of when the plugin received the gaze
- A list of IDE context data generated from a gaze event

- The x and y pixel coordinate (duplicated from the gaze event)
- The name, extension, and path of the file that was viewed
- The line and column in the file that was viewed
- The size of the font being used in the editor

The IDE plugins allow a participant to do many things that would normally not be allowed during an eye-tracking study: scrolling, opening new files, and using built-in editor tools like find.

3.1.3 iTrace-Chrome

In addition to supporting development environments, iTrace supports conducting eye-tracking research within the Google Chrome web browser. iTrace-Chrome, a Google Chrome plugin, connects to iTrace-Core in the same way that the IDE plugins do and maps received gaze data to elements on the web page. This mapping works similarly to that of the IDE plugins, meaning the participant can scroll around the webpage. iTrace-Chrome supports tracking the following webpages: Google Search results, Stack Overflow Questions, Stack Overflow search, GitHub Issues lists, GitHub profiles, GitHub Pull Requests, and Bugzilla Bug reports.

3.1.4 iTrace-ScreenRecord and the OBS Plugin

When conducting eye-tracking studies on dynamic stimuli, researchers will often want to record the stimuli during the session. This could be to map eye-tracking data back to the stimulus, to create visualizations of the session, or to review a session to better understand what a participant did. Additionally, many researchers want to record additional

information, such as a live feed of the participant's camera, for additional research/evaluation. When recording such information, it is important to ensure that the recording length matches the length of the eye-tracking session to ensure accurate mapping between the data. Typically, recorded videos are manually adjusted to match the eye-tracking data timeline-wise, but this is a very manual process and can require a lot of effort – especially for studies with many participants.

To facilitate accurate recording matching, iTrace offers a plugin for the Open Broadcaster Software (OBS). OBS is an opensource tool meant for recording/streaming videos from a desktop computer. The plugin, named iTrace-ScreenRecord [Behler, Chiudioni, Ely, Pangonis, Sharif, and Maletic 2023], receives session start and end signals from iTrace-Core to trigger the starting and ending of a recording [Behler, Chiudioni, Ely, Pangonis, Sharif, and Maletic 2023]. This ensures that the resulting video will be the same length as the eye-tracking session. OBS also supports video overlaying, meaning artifacts like a live camera feed, text, and other visual markers can be added to a video. Figure 3.2 showcases an example of a video recorded by OBS and iTrace-ScreenRecord during an eye-tracking session.

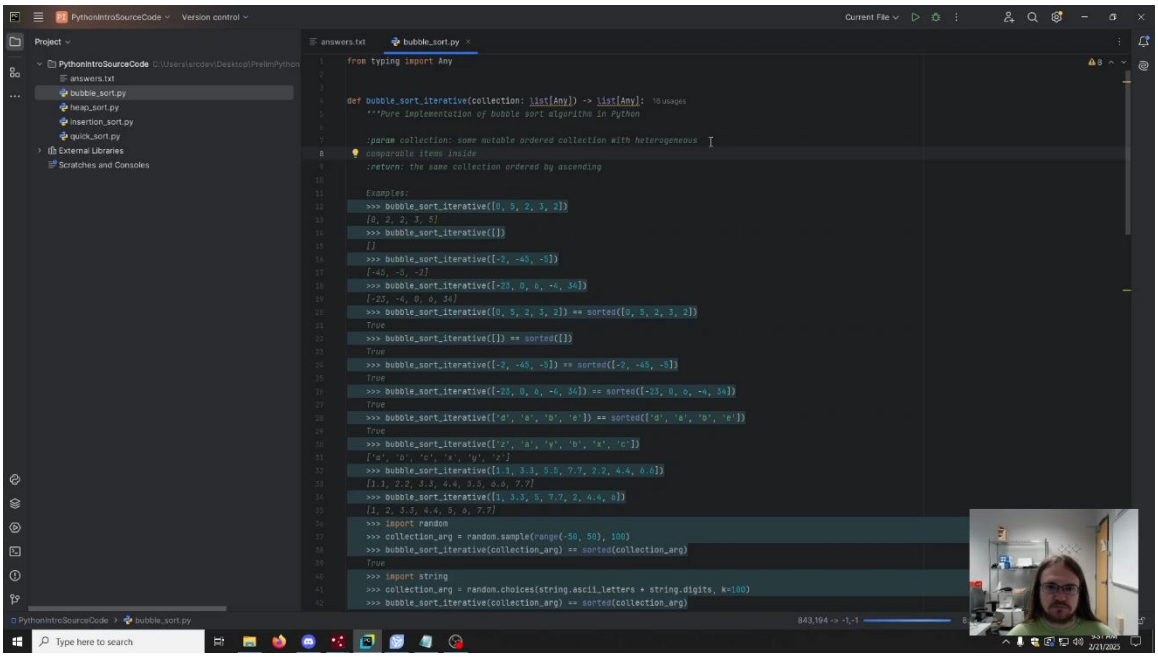


Figure 3.2. A frame from a video recorded by iTrace-ScreenRecord. The bottom right of the video also has a live feed from a webcam overlayed, allowing for additional processing of the participant's face

3.1.5 Deja Vu

A limitation with iTrace's real-time mapping of gaze data to the IDE stimulus is processing speed. Eye-trackers range in their collection rates. Typical research-grade eye-trackers collect data at 120-300 Hz, whereas higher-end eye-trackers can reach 1000+ Hz. Because the data mapping is managed by a plugin running as a subprocess of an editor, the mapping cannot always handle higher data rates without data loss. To support these high-rate eye-trackers, iTrace-Core supports a data-replay mechanism. This mechanism, called Deja Vu, is implemented in two main parts [Zyrianov, Peterson, Guarnera, Behler, Weston, Sharif, and Maletic 2022]. The first part, Deja Vu Record, is enabled during an eye-tracking session. Deja Vu Record records all eye-tracking gaze information sent by the eye-tracker, as well as all mouse and keyboard inputs. This data is output into a comma-separated values

(.csv) file. The second part, Deja Vu Replay, uses the .csv file to replay all the eye data while also virtually pressing the same keys and moving the mouse cursor exactly as they were during the eye-tracking session. A scaling factor can slow this replay, so data that was previously too fast for a plugin to capture can now be processed in time.

3.1.6 srcML

While not a part of the iTrace Infrastructure, srcML[2] [Collard, Decker, and Maletic 2011; Collard, Decker, and Maletic 2013] is a vital part of iTrace's data collection pipeline. srcML is an XML format for source code that enables exploration, analysis, and manipulation. Because it is an XML format, any XML-aware tool can analyze the source code. Critical for iTrace's ability to map file positions to specific tokens, srcML also supports the labeling of each token's starting and ending line and column information.

3.2 Post-Collection Processing

iTrace additionally assists in the post-processing of collected eye-tracking data by providing tools to aid researchers in carrying out various tasks and analysis.

3.2.1 iTrace-Toolkit

After collecting data from an eye-tracking session, the data needs to be processed before analysis. The analysis required by a researcher will depend on what research they are trying to do. To aid researchers in their work, iTrace offers a post-processing tool for common analysis tasks. iTrace's post-processing tool is known as iTrace-Toolkit [Behler,

[2] See https://www.srcML.org/

Weston, Guarnera, Sharif, and Maletic 2023]. iTrace-Toolkit provides three major features for researchers: mapping eye-tracking data to syntactic tokens in source code, calculating fixations and saccades, and querying the calculated fixations. Figure 3.3 shows the GUI of the iTrace-Toolkit.

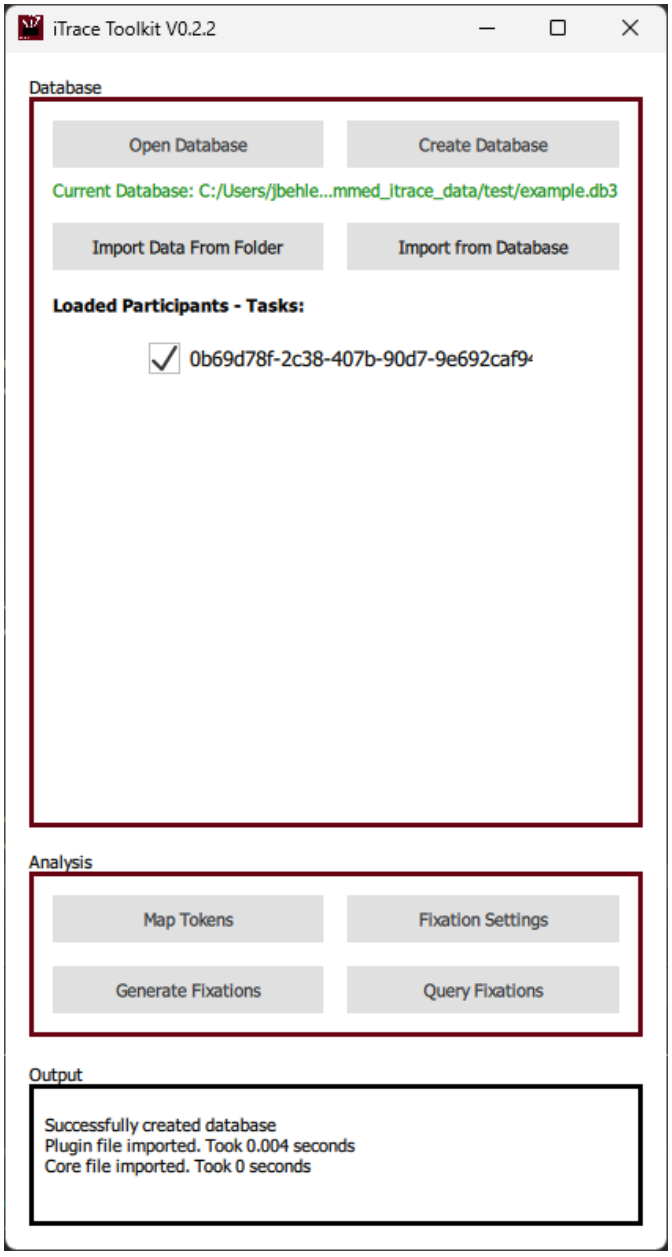


Figure 3.3. The GUI of iTrace-Toolkit with a single eye-tracking session loaded

Token mapping is the process of mapping eye-tracking data collected during an eye-tracking session to the source code viewed. As mentioned previously, the iTrace IDE Plugins will report the line and column in the source code file that were viewed. This is already a form of mapping; however, the IDE plugin only reports the line and column that was viewed – essentially just reporting the character that was viewed. The plugin does not report any information about what that character means within the greater context of the code. Researchers working with eye-tracking for software engineering do not just want to know that a participant viewed a specific character in the file – they want to know if the participant viewed a particular function, or a particular variable, etc. iTrace-Toolkit calculates this data using the data from the collected plugin data files and a srcML file of the stimulus source code. This process records the following data for each gaze collected:

- The full token that was viewed (not just the character)
- The syntactic context of the token
- A unique XPath that points to the token within the srcML file

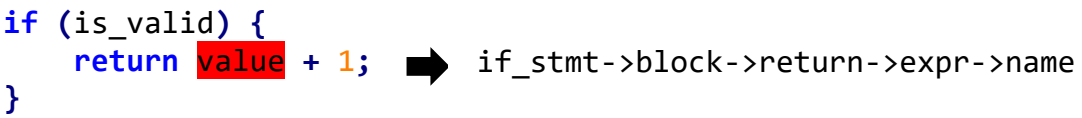


Figure 3.4. An example of a syntactic context of a token. The target token, `value` (highlighted red), has the syntactic context of being a name in the expression of a return statement within the block of an if statement

The syntactic context of a token is its syntactic meaning and its hierarchical position within the source code. Specifically, the syntactic context is the token's lineage within the srcML markup (see Figure 3.4). This information allows researchers to identify not just which specific characters/tokens have been viewed, but also how many times a collection

of tokens within a single syntactic feature has been viewed, and to answer any other questions related to specific language features. It is important to note that iTrace-Toolkit's token mapping is only available for eye-tracking studies done on languages that srcML supports (at the time of writing, C, C++, C#, Java, and Python). The other features of iTrace-Toolkit are not limited to a specific language, but any token mapping will require a separate implementation from a researcher.

The next feature offered by iTrace-Toolkit is the ability to map gaze data to fixations and saccades. The human eye moves in two distinct ways – small, jittery movements while the eye focuses on a specific area, and quick, scanning jumps between those foci. Those types of movements are known as fixations and saccades, respectively. Fixations and saccades are vital for eye-tracking research, as they have been shown to indicate participants' visual effort during tasks [Sharafi, Shaffer, Sharif, and Guéhéneuc 2015]. Eye-trackers cannot directly capture these movements – they capture instantaneous snapshots of where the eye is looking at a specific point in time. To best understand a participant's eye movements, the gazes need to be converted into fixations and saccades. Many algorithms exist to convert gaze points into fixations and saccades [Andersson, Larsson, Holmqvist, Stridh, and Nyström 2017]. iTrace-Toolkit implements the Olsson BASIC, I-DT, and I-VT algorithms for calculating fixations and saccades [Olsson 2007; Salvucci and Goldberg 2000].

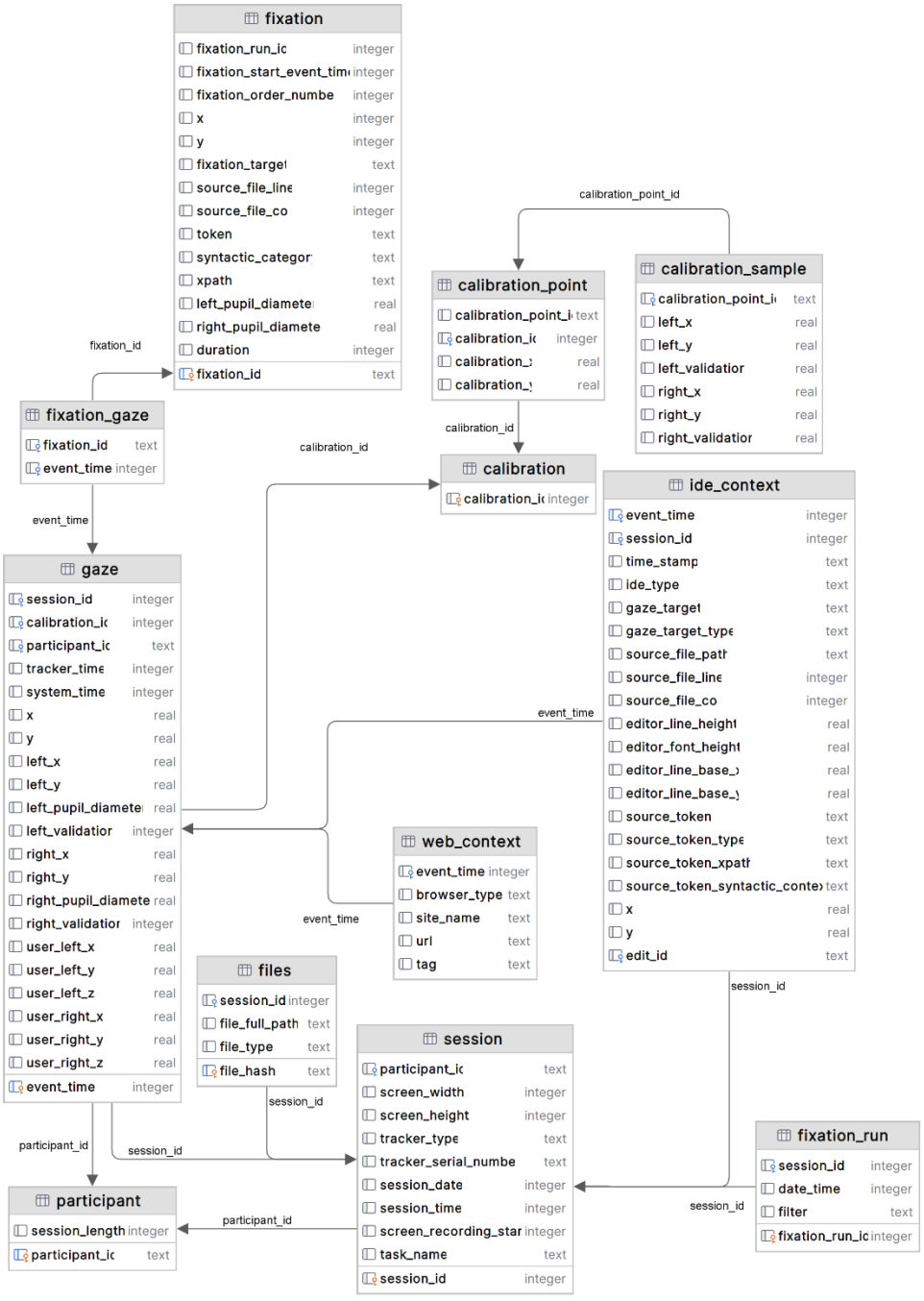


Figure 3.5. The Entity Relationship Diagram of the iTrace database file

iTrace-Toolkit saves all the data it receives as input and all the data it produces into a SQLite database file, known as the *iTrace database file*. Figure 3.5 describes the schema

of the iTrace database file using an entity-relationship diagram (ERD). The database is a collection of data from all parts of the iTrace Infrastructure, including duplicate values from the core and plugin files and all output from iTrace-Toolkit. iTrace-Toolkit supports processing multiple eye-tracking sessions simultaneously and will even save multiple runs of different fixation algorithms to allow for easy comparing and contrasting. Because the data is stored in SQLite format, querying it is simple. iTrace-Toolkit also supports some simple filtering options to make gathering desired data even simpler.

iTrace-Toolkit is currently under development, with many new features being planned. Support for saccades is currently underway and nearly complete. Work by Guarnera et al. that supports syntax-aware fixation correction is being implemented into iTrace-Toolkit as a part of the fixation calculation process [Guarnera, Behler, Sharif, and Maletic 2025].

3.2.2 iTrace-Visualize

Eye-tracking generates a lot of data. A single five-minute eye-tracking session with a basic eye-tracker running at 120 Hz will result in over 30,000 gaze points and approximately 1,000 fixations. Running that session on ten participants means a researcher will have 300,000 gaze points and 10,000 fixations to analyze. Eye-tracking data is also difficult to interpret after collection, as there are many data points to consider. The sheer volume of data and its complexity make eye-tracking data difficult to understand at a glance. Researchers want to understand what a participant was looking at during a session; whether they are assessing their experimental design or examining an outlier, they need to understand their data. Data visualization is a common method for understanding collected

eye-tracking data. Many eye-tracking studies use visualization techniques such as heatmaps and gaze plots to show where a participant was looking at a stimulus. Figure 3.6 showcases some examples of these visualizations.

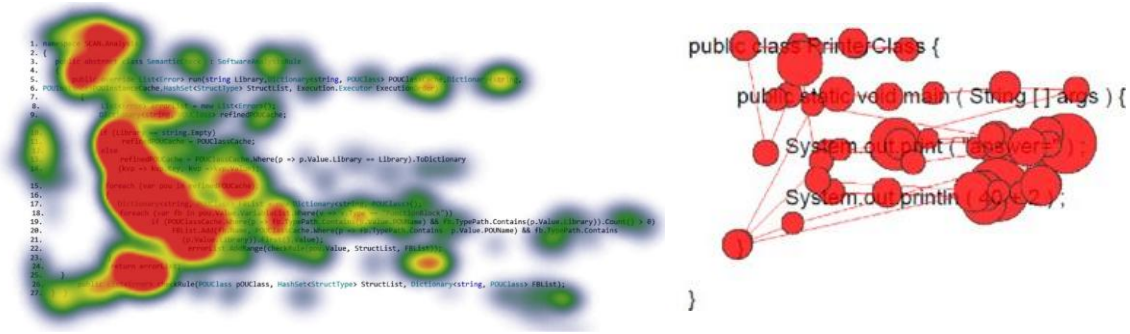


Figure 3.6. Examples of a heatmap (left, [Chandrika and Amudha 2025]) and gaze plot (right, [Sharafi, Sharif, Guéhéneuc, Begel, Bednarik, and Crosby 2020]) on static source code stimulus

However, traditional data visualization techniques do not work well for iTrace. Overlaying information over the stimulus is ineffective when applied to the dynamic IDE stimulus in iTrace. At any second, the stimulus could look completely different, and different regions of the screen could be important. This makes heatmaps and gaze plots confusing to use. To provide useful visualizations to its users, iTrace offers a tool for generating them. The tool, named iTrace-Visualize, can produce three types of visualizations that suit different comprehension needs for researchers [Behler, Chiudioni, Ely, Pangonis, Sharif, and Maletic 2023; Behler, Villalobos, Pangonis, Sharif, and Maletic 2024].

The first visualization iTrace-Visualize offers is a marked-up video of the eye-tracking session. Using a video recorded with iTrace-ScreenRecord and an iTrace database containing the post-processed data from that session, the eye-gaze and fixation data can be

overlaid on the video to play in real time. The marked-up eye data slowly fades away over time, so the entire video does not become cluttered with it, and the data timeline is preserved throughout the video. iTrace-Visualize's video output is analogous to a standard gaze-plot, as both show the gaze data timeline over the stimulus. Figure 3.7 shows an example of the video output.

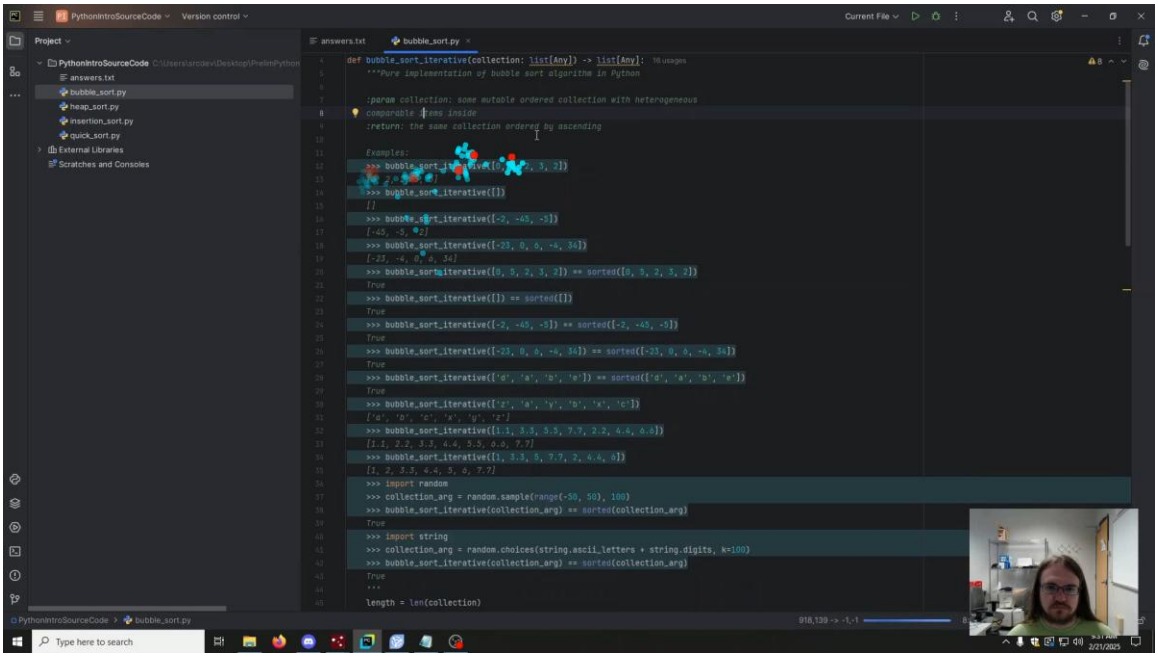


Figure 3.7. A frame from an example marked-up video from iTrace-Visualize. The eye-tracking session depicted is the same session from Figure 3.2

The second visualization iTrace-Visualize offers is tokenized heatmaps. Visualizing heatmaps over a video does not work well, as the heatmap can become outdated as soon as the participant scrolls or opens a new file. The heatmap could fade away over time, similarly to how the marked-up video is done, but that does not provide any new information that the marked-up video does. Instead, iTrace-Visualize can create a specialized heatmap that shows which tokens in the source code were viewed most. Using

srcML as input, the tool creates an image representation of the source code, coloring all tokens according to how many times a participant viewed them. Figure 3.8 shows an example heatmap produced by iTrace-Visualize.

Figure 3.8. An example of a tokenized heatmap made by iTrace-Visualize. The color legend in the bottom right indicates the number of fixations on a given source code token

The third visualization iTrace-Visualize makes is Region of Interest (ROI) Scarf Plots. Sometimes, researchers are not interested in which specific tokens a participant viewed, but rather which region of the code was viewed. For example, an if statement within the code stimulus could have a logic error, and the researcher wants to know how long the participant spent looking at that if statement and when they first viewed it. iTrace-Visualize allows a researcher to define specific lines of code as ROIs and then generate a

scarf plot that shows the timeline of which ROIs a participant was viewing across the whole session. Figure 3.9 shows an example scarf plot made by iTrace-Visualize.

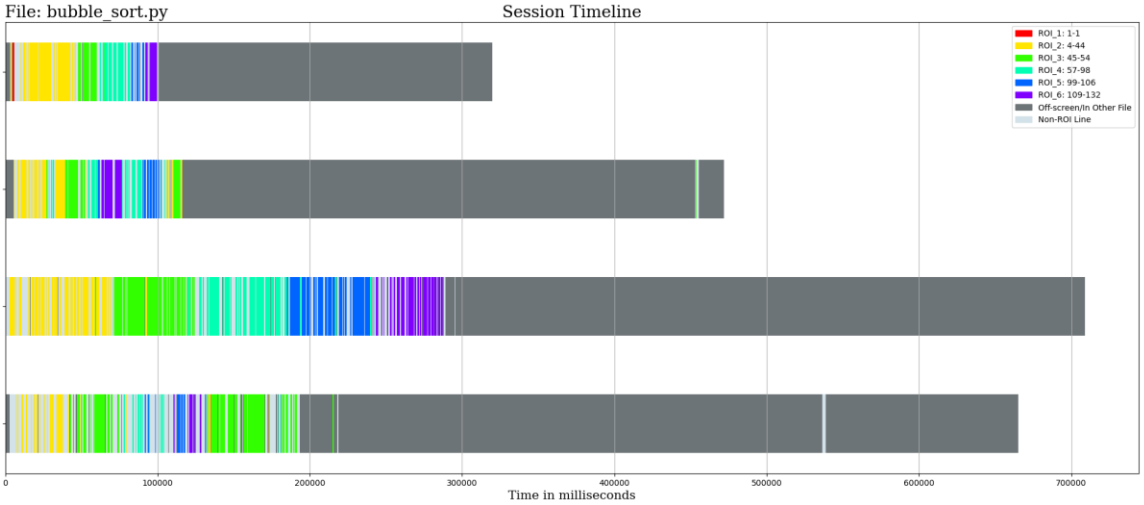


Figure 3.9. An example of a region-of-interest scarf plot generated by iTrace-Visualize. This scarf plot shows four participants who all viewed the same file, but at different times

CHAPTER 4

Source Code Edits

While programming, developers perform various actions that modify the text of source code files. This work uses specific definitions for how source code is edited and changed at different levels of granularity. Informally, a *text event* is a hardware event that results in some addition or removal from the source code. It includes the hardware inputs (keypress/mouse click) and the timestamp of the event.

Text events are collections of specific characters that are added/removed from the code, the cursor position(s) the characters are affected at, the timestamp of the event, and what hardware inputs are responsible for the text event occurring. A text event is the direct consequence of some computer input or interaction and thus cannot be divided into a smaller unit of change. One or more text events make up larger modifications to the source code. Informally, a *code edit* (*edit*) is a single semantic change made by a developer while they are actively working on the file.

Code edits are comprised of one or more text events, done by the developer, and typically represent additions or removals from the source code text. Figure 4.1 showcases an example of a function being renamed, which is an example of a code edit.

Original

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean linearNumberSearch(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (linearNumberSearch(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

Modified

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean isNumberInList(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (linearNumberSearch(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

Figure 4.1. the srcDiff representation of a single function being renamed

A *code change* (*change*) is the delta, as computed by a differencing algorithm (e.g., diff), between two revisions or commits of a file. A code change can be made up of many edits, and different sequences of edits can lead to the same change (i.e., deleting a word and adding a word vs. adding a word and deleting a word). Figure 4.2 showcases an example of a code change – a code commit.

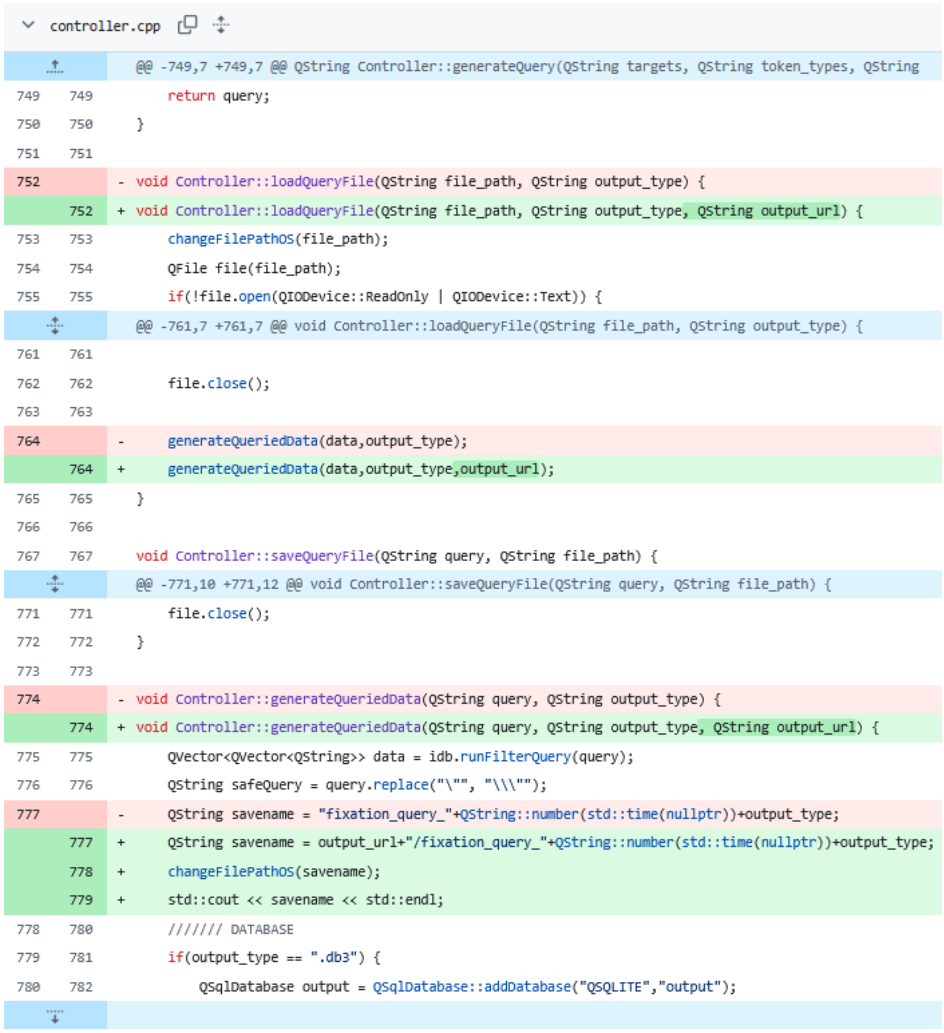


Figure 4.2. The diff representation of a commit to a git repository

4.1 Hardware Inputs and Timing

Using a mouse and keyboard, programmers edit source code files. Hardware inputs are how programmers interact with the computer and ultimately cause changes to the text of source code and are thus important to defining text events and code edits. Table 4.1 lists common hardware inputs used to affect source code.

Table 4.1. A list of common types of hardware inputs that are used to edit source code. This list is not exhaustive, and some hardware inputs might not be available depending on the development environment a developer uses

Hardware Input	Name	Change to Text	Description
Character key	Insert Character	Insertion	A single character key on the keyboard is pressed, inserting a single character
Backspace key	Backspace	Deletion	The backspace key is pressed, deleting the character before the cursor position
Delete key	Delete	Deletion	The delete key is pressed, deleting the character at the cursor position
CTRL+V	Paste	Insertion	Text previously copied into the computer's clipboard is pasted into the source code
CTRL+X	Cut	Deletion	Highlighted text is deleted and copied to the computer's clipboard
Highlight+Any insert action	Text replace	Replacement	A selection of characters is highlighted and then replaced with new text on any insertion action
Highlight+Any delete action	Area delete	Deletion	A section of characters is highlighted and then deleted
Highlight+Mouse drag	Move text	Deletion/Insertion	A selection of characters is highlighted and then dragged to a new position using the mouse
Highlight+CTRL+Mouse drag	Copy text	Insertion	A selection of characters is highlighted, CTRL is held, and then copied to a new position using the mouse
CTRL+H	Find and Replace	Replacement	Built-in editor tool – replaces all instances of one string with another

Hardware inputs vary widely in their ability to affect the underlying text. Some insert code, others delete code. Complex inputs can result in text moving, being duplicated, or text being inserted in multiple places at once. Some modern development environments support multi-cursor functionality, allowing hardware inputs to cause changes to the source code text at multiple text positions simultaneously.

Alongside the hardware inputs, timing is an important part of what makes an edit. If a developer types 20 characters in a row, there is most likely a significant difference in the semantic meaning of the inputs if there is no pause between them versus if there is a long, 30-second pause between the tenth and 11th character. Timing between inputs also varies between developers who have different typing skill levels and are working in different environments.

4.2 Types of Edits

There are numerous ways to categorize source code edits. Table 4.2 details the edit categories used in this work. These edits are chosen because they provide a semantic-free representation of text changes in source code and enable future work on building new, semantic-focused categories. These categories also help clarify changes by avoiding complex issues when dealing with syntax changes in source code (for example, encapsulating multiple lines of code in a comment).

Table 4.2. The categories of edits used in this work

Category Name	Description
insertToken	A new source code token is inserted into the code
insertTokens	Multiple new source code tokens are inserted into the code
deleteToken	An entire token within the source code is deleted
deleteTokens	Multiple tokens within the source code are deleted
replaceToken	A source code token has its text changed/replaced
replaceTokens	Multiple source code tokens have their text changed/replaced
whitespaceFormatting	Only whitespace is inserted/deleted

4.3 Formal Definitions

With the understanding of the makeup of text events and edits, formal definitions can be introduced.

Let Σ denote a finite alphabet of characters. A source code artifact is modeled as a string $S \in \Sigma^*$, where Σ^* represents the set of all finite sequences over Σ. In addition, let H denote the set of all possible hardware interactions, including keyboard key presses, mouse interactions, and editor-level actions. These interactions serve as the observable triggers for modifications to the source code.

A ***text event*** is defined as the tuple $e = (t, h, p, \delta)$, where t is a timestamp drawn from a totally ordered set of time values, $h \in H$ is the hardware interaction responsible for the event, p represents the position or range of positions within the source string at which the event occurs, and δ is an operation describing how the string is modified.

The operation δ captures the fundamental ways in which the source string can be modified. In this work, δ is defined as one of the following: an insertion operation $\mathrm{insert}(s)$, where $s \in \Sigma^*$ is a string to be inserted; a deletion operation $\mathrm{delete}(k)$ where $k \in \mathbb{N}$ is the number of characters to remove; or, implicitly, a replacement operation which can be expressed as a composition of a deletion followed by an insertion – formally, $\mathrm{replace}(s, k) \equiv \mathrm{insert}(s) \circ \mathrm{delete}(k)$, where $s \in \Sigma^*$ and $k \in \mathbb{N}$. These operations are sufficient to model all text modifications observed in typical code editing environments.

To formally capture the effect of a text event on source code, each event is associated with a transformation function $[\![e]\!]: \Sigma^* \to \Sigma^*$. Given an input string S, the function $[\![e]\!](S)$ produces a new string S' reflecting the result of applying the text event's

operation at the specified location(s). For example, with an initial source code string expressed as $S = x \cdot y$ and an edit e representing an inserting of string s at position i with $i = |x|$, applying e to S results in a string $S' = x \cdot s \cdot y$. Similarly, a deletion of k characters at position i removes a substring of length k from S, yielding a new string formed by concatenating the remaining prefix and suffix.

This formulation allows text events to be interpreted as atomic transformations over source code strings. Therefore, ***code edits*** can be modeled as a sequence of such transformations. Given an initial source code string S_0 and a sequence of text events $e_1, e_2, \ldots, e_n$, the resulting sequence of program states can be defined as $S_i = [\![e_i]\!](S_{i-1})$, for $i \in [1, n]$, where each text event induces a transformation on the source code state.

While text events represent the finest-grained units of modification, software development activity is typically structured around higher-level units of change. In this work, a code edit is defined as a contiguous subsequence of text events that collectively correspond to a single logical and semantic modification performed by a developer during active editing.

Formally, let $E = \langle e_1, e_2, \ldots, e_n \rangle$ denote the ordered sequence of text events associated with a source code file across a development session. A code edit d is defined as a subsequence of text events $d = \langle e_k, e_{k+1}, \ldots, e_{k+m} \rangle$ where $1 \leq k \leq k + m \leq n$. The effect of a code edit is defined as the composition of the transformation functions induced by its constituent text events:$[\![d]\!] = [\![e_{k+m}]\!] \circ \cdots \circ [\![e_k]\!]$.

In practice, the boundaries of a code edit are not directly observable and must be inferred from the event stream. As a result, edit segmentation is inherently ambiguous and

may vary depending on the grouping heuristic applied and the opinion of the person defining the bounds of the edit.

Lastly, a ***code change*** is defined as the delta result from applying a differencing algorithm over two distinct versions of a source code artifact. Let $S_a, S_b \in \Sigma^*$ denote two snapshots of a program state. A code change is then defined as $\Delta(S_a, S_b) = \text{diff}(S_a, S_b)$, where diff is a function that computes a finite set of transformations required to convert S_a into S_b. Different differencing algorithms will produce a different set of transformations, but they ultimately all describe the same code change, $S_a \Longrightarrow S_b$. The set of these transformations are referred to as an edit script in differencing work and are analogous to code edits in this work. However, the code edits in this work examine human editing behavior, while the transformations from differencing algorithms do not model human behavior.

In contrast to text events and code edits, which are defined over fine-grained, interaction-level modifications, a code change operates at a coarser granularity and is typically derived from versioned representations of source code, such as commits in a version control system. Conceptually, a code change can be interpreted as a sequence of code edits that, when composed, transform an initial program state into a later one. That is, there exists a sequence of edits $d_1, d_2, \ldots, d_m$ such that $S_b = [\![d_m]\!] \circ \cdots \circ [\![d_1]\!](S_a)$. However, in traditional software engineering practice, a code change is primarily defined syntactically as the output of a differencing operation between two versions of a file or repository state, without explicit reference to the underlying sequence of developer actions

that produced it. As such, code changes do not inherently encode semantic intent but rather represent structural differences between snapshots of the program.

Higher-level artifacts can be further constructed by aggregating multiple code changes over time, such as branches, pull requests, or release versions. These structures represent development activities over long periods of time and across (potentially) multiple developers; however, they are not further formalized in this work.

CHAPTER 5

Collecting Data on Developer Code Editing

To answer questions on how source code is edited, real data of developers editing source code is needed. Work by Brown et al. has previously collected code-editing information from users of the BlueJ Java IDE application [Brown, Kölling, McCall, and Utting 2014; Brown, Altadmri, Sentance, and Kölling 2018]. However, as of January 2026, this dataset has been made unavailable to the public, meaning there is no accessible dataset of editing data. To acquire this data, a survey is used to collect developer skill and demographic information, along with real code-editing data. The survey is fully hosted in a web browser on a GitHub Pages website. Survey data is sent directly to a machine within the Kent State Computer Science Department. The survey is aimed at developers with college freshman-level or higher programming experience. It is distributed among Kent State Computer Science students, students at other Universities, and some industry developers.

5.1 Survey Design

The survey is broken into three main parts: the pre-questionnaire, the typing tasks, and the post-questionnaire. The questionnaires are simple demographic questions to collect relevant information on programming skills, detailed below. Of the typing tasks, there is

one prose task and ten programming tasks. The programming tasks are short, simple problems designed in an open-ended way to capture real, authentic editing habits from developers. The programming tasks are offered in three languages: C++, Java, and Python. The prose task involves manually copying a sentence of text by typing. While the participant completes the typing tasks, data is collected from their mouse and keyboard inputs. Three data streams are concurrently collected:

1. Any keyboard and mouse inputs made within the browser tab that is hosting the survey
2. The contents of the text entry box that contains the source code
3. The current position of the text cursor and any highlighting done to text within the text entry box

Unfortunately, standard survey collection platforms lack the infrastructure to collect this kind of information. To make conducting the survey possible, a custom, web-based survey tool was created. The survey tool is comprised entirely of simple HTML, JavaScript, and CSS, meaning it works on any standard operating system and web browser. The tool is hosted publicly on a GitHub Pages webpage, making distributing the survey as simple as distributing a URL. All data collection occurs entirely within the webpage when a participant opens the webpage, meaning no information is collected outside the browser or in other tabs. The entire tool is client-based – the only communication to a server is when the collected data is sent at the end of each task. All collections are done anonymously.

5.2 Survey Questions

The pre-questionnaire asks the following demographic questions:

- "What is your age group?"
- "What is your gender?"
- "Please list all programming languages you have worked with before / feel comfortable working with"

The post-questionnaire asks the following demographic questions:

- "What is your highest education level?"
- "Are you currently in school or higher education? If so, what is your current class rank?"
- "How many years experience do you have working professionally as a programmer?"
- "What do you rate your level of experience and skill in programming?"

The post-questionnaire questions are asked after the programming tasks are completed to avoid making participants feel judged for the accuracy of their responses, which could affect how they complete the tasks.

For each programming task, participants are given a prompt and a text entry box with starting code. The participant can freely modify the text in the entry box, and common programming shortcuts (such as using the Tab key to indent code) have been implemented to make editing easier. Table 5.1 lists the prompts for the typing tasks as well as the reasoning for their inclusion in the survey. APPENDIX A details the code displayed at the

start of each task. Before starting the survey, participants are told that they can submit no wrong answers, and to complete the tasks to the best of their ability.

Table 5.1. All typing tasks asked of survey participants, and the primary reason for their inclusion in the survey

Typing Task Prompt	Notes
Duplicate the following text manually - do not use copy+paste: Software engineering is a branch of both computer science and engineering focused on designing, developing, testing, and maintaining software applications.	Establishes a baseline for typing speed and editing behavior using a prose task
Change all the variables named `x` to be `width` and the variables named `y` to be `height`.	Evaluates text replacement and cursor movement through a simple name change task
The function `calculateFinalScore` calculates a student's score in a course. The course is removing the bonus assignment, and the function needs to be updated. Remove all code that involves computing the bonus.	Evaluates strategies for text deletion
The `calculateDailyRevenue` function calculates the per-day revenue gained by a delivery company. The function is going to be expanded to allow the optional calculation of potential revenue if no orders were cancelled. Add a new boolean parameter to the function that allows for this upgrade to be made.	Evaluates insertion behavior by requiring addition of tokens within dense code regions
Rename the function `foo` to have a more appropriate name.	Evaluates naming behavior by requiring participants to interpret a function and iteratively construct an appropriate name
Write a comment above the `isStrongPassword` function that explains what it does along with the criteria for a strong password.	Compares prose writing behavior within a code context using a comment-based task
Convert the for loop in the following code to be a while loop.	Evaluates code restructuring through tasks requiring multiple token additions and modifications
Create a new class called `Log` and encapsulate the following 3 functions into it as its methods. Make the `log` parameter a data member of the class and remove it from the parameter lists of the methods.	Evaluates code movement and reorganization during code restructuring
Convert the for loops in the following code to be while loops.	Evaluates complex restructuring behavior through a multi-step modification task
The following code contains two functions that average students' exam and homework scores. Replace the two specific functions with one general function that can work on both the exam and homework arrays. Move the printing to be in the main function.	Evaluates code reuse behavior through the consolidation of functionality from multiple locations
Implement the `readAndSumNumbers` function. It should read two numbers from the provided file, print the sum of the two, then return the value.	Evaluates code creation by requiring participants implement a simple function from scratch

5.3 Survey Responses

The survey was conducted over a period of 34 days (April 6th - May 9th 2026). The survey was distributed to two main groups: university students and industry practitioners. For students, the survey was sent out to various computer science classes across many universities, some offering bonus points for its completion, and to various school clubs and forums. For practitioners, the survey was sent to the author's industry connections and distributed by those connections to other employees at their companies. In total, 243 participants filled out all or part of the survey. For each task and questionnaire completed by a participant, a data file is created, resulting in a total of 3,150 data files.

Out of the 243 participants:

- 219 completed all tasks and both questionnaires
- One completed all tasks and the pre-questionnaire, but not the post-questionnaire
- 18 completed the pre-questionnaire and some of the tasks, but not all
- 5 completed only the pre-questionnaire

Additionally, of the 243 participants, 13 submitted multiple responses to some tasks and questionnaires, and 11 submitted multiple responses to all tasks and questionnaires. The exact reason why these participants submitted multiple responses is not completely known, but some possible reasons why include multiple individuals using the same computer within a short time period of each other, someone accidentally closing the survey and restarting it, or a student participant forgetting to screenshot their participant ID and submit it for bonus points in their class. When a task has two or more responses from the

same participant, the response with the most text events is retained, and the others are discarded. 159 task responses are discarded through this criterion.

Not all responses to the survey contained useful information. While responding, some participants did not make any meaningful edits to the task's source code. Some participants either made no changes to the source code for a task or made only one or two simple edits before submission. In a few cases, some participants' only change to the code was replacing the entire text with the completed version in a single paste, implying they either completed the task in an external editor and pasted the result or used an LLM to complete the task for them. 204 task responses have three or fewer text events recorded and are discarded for not containing enough information for any meaningful analysis.

After these two discards, there are four participants with only the prose task remaining. Since there is no qualifying code-editing information to study from, their remaining prose tasks are also discarded.

In total, 367 task responses are discarded from further analysis. With the removal of these responses, the total number of participants is 235, and the number of data files collected is 2,741.

The following is demographic data collected from the survey. Note that not all of the 235 participants completed both the pre- and post-questionnaires, meaning that no set of demographic data will total 235.

170 participants identified as male, 53 as female, six as nonbinary or another gender, and five declined to disclose their gender.

Most participants are bachelor's students currently enrolled in college. Figure 5.1 and Figure 5.2 showcase the participants' education levels.

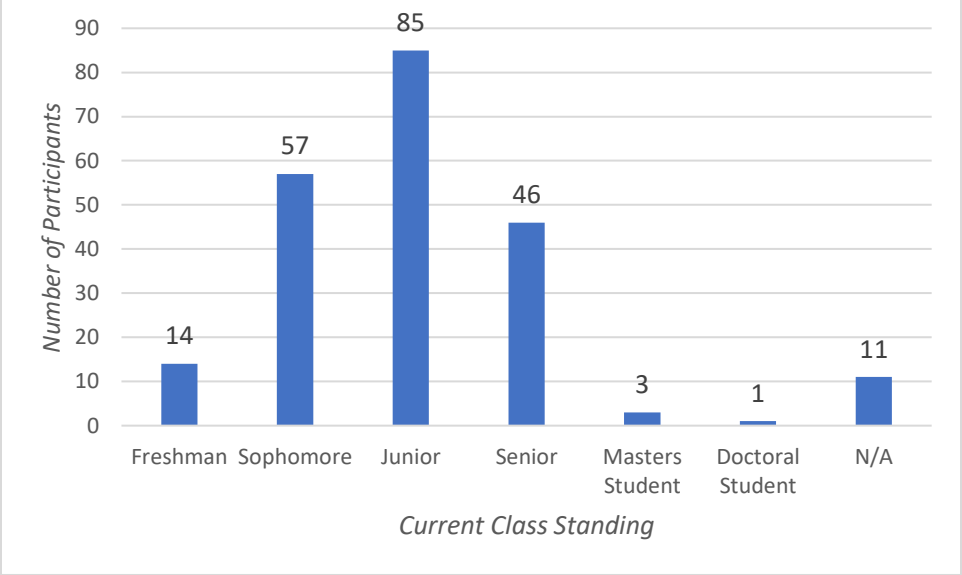


Figure 5.1. The participants' class standing at the time of their response. The N/A bar represents participants who are not currently enrolled in any education

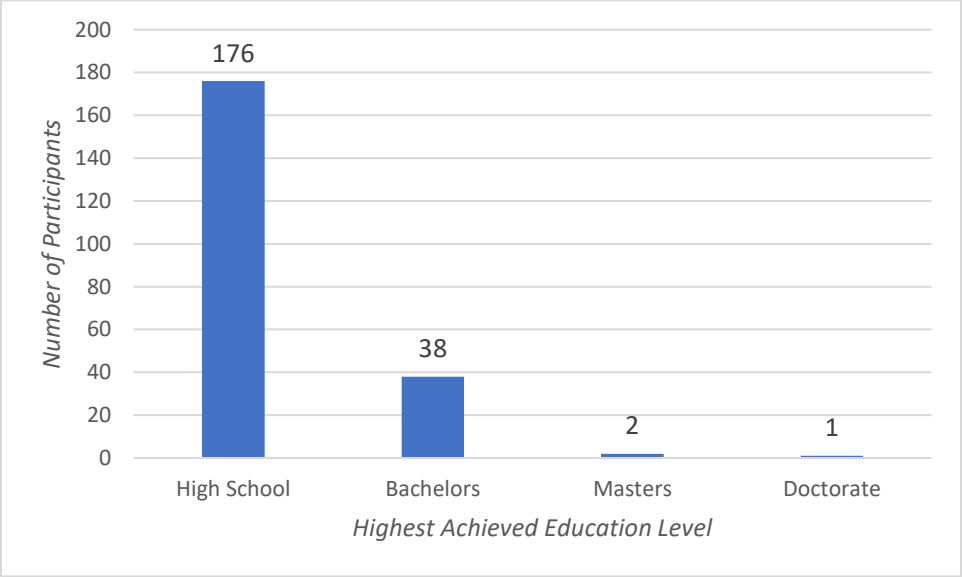


Figure 5.2. The highest achieved education level of the participants at the time of their response

It is important to note that 27 participants report having completed a bachelor's degree while also reporting being currently enrolled as undergraduates (freshman to senior class standing). These responses are likely incorrect responses due to confusion caused by a misunderstanding of the demographic questions or correct responses from students who already have a degree and have returned to school for further education.

Figure 5.3 details the professional experience of participants. Most participants have had little experience programming professionally, and only a small minority have had more than 3 years of professional programming experience. Despite this, most participants report being of an intermediate level of skill in programming, as shown in Figure 5.4. Most participants who are not enrolled in college and have many years of industry experience also self-reported an intermediate skill level.

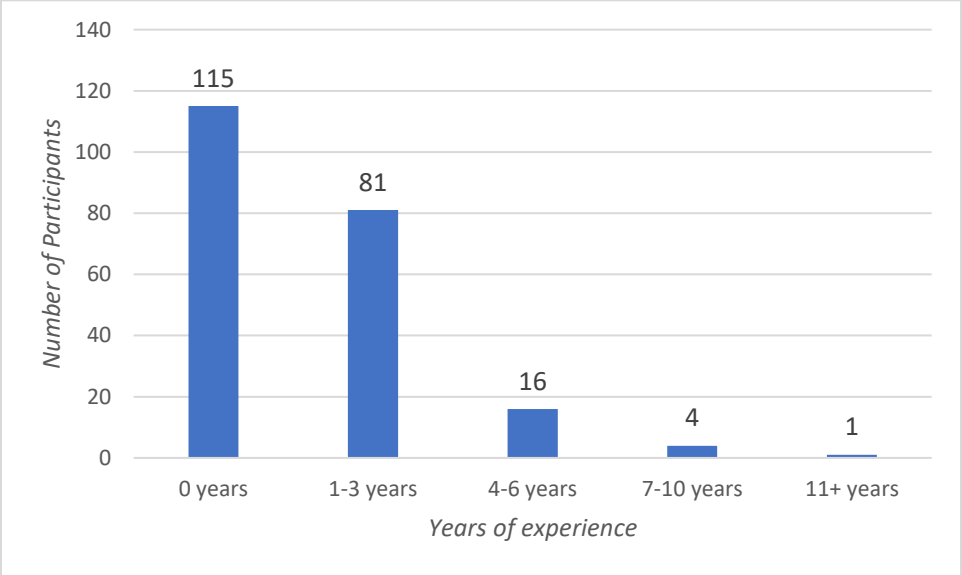


Figure 5.3. Reported number of years of experience working professionally as a programmer by participants

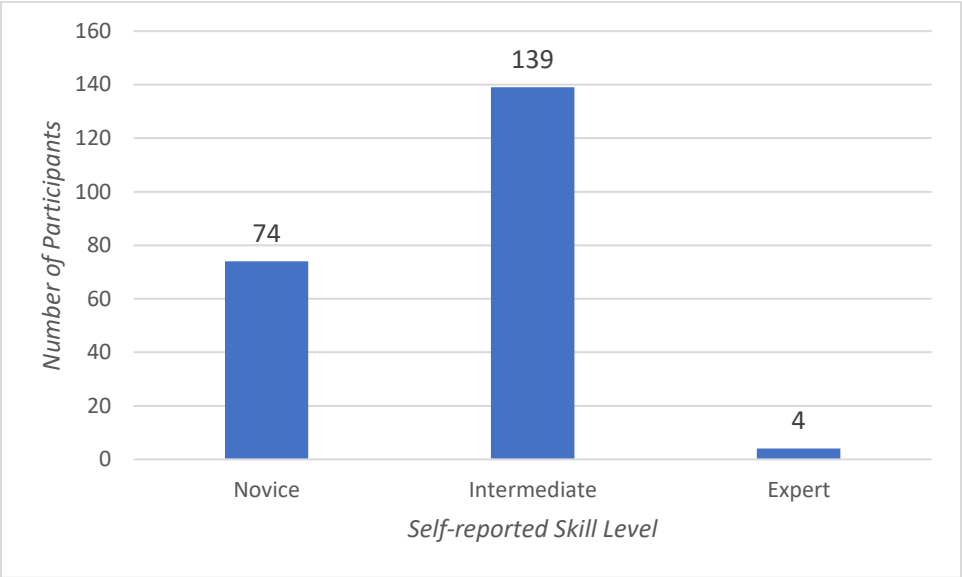


Figure 5.4. Self-reported evaluation of participants' skill at programming

CHAPTER 6

Analysis of Developer Editing Behavior

Having established the survey's design and implementation, this chapter focuses on the analysis of the collected data, specifically to address RQ1. The following two sections provide an overview of the collected dataset and what data is used for further analysis. Then, the chapter explores many facets of developer behavior during code editing through empirical analysis of participants' responses.

6.1 Collected Programming Task Data

In total, 120.4 hours of editing are collected from the 235 participants across 2,060 programming task responses. This means each participant spent an average of 3.5 minutes per task. The longest amount of time spent completing a programming task by a participant is 18.8 hours, and the shortest is 6.8 seconds. Of the 235 participants, 139 completed tasks in C++, 86 completed them in Python, and seven completed them in Java.

During each programming task, three categories of information are collected: hardware input events, text events, and selection events. Together, these events provide a comprehensive representation of the programming task and enable analysis of developer behavior while editing source code. In the following three subsections, RQ1.1 is explored

by examining these events, revealing the interactions and inputs developers use to create their changes to the source code.

6.1.1 Hardware Input Data

There are two types of hardware information collected during a programming task: mouse interactions and keyboard key presses. Of these, the survey program captures six specific events. In total, 646,736 hardware events (79,526 mouse events and 567,210 key events) are collected from survey participants. All hardware events have an associated timestamp. Table 6.1 details the six kinds of hardware input events captured by the survey tool.

Table 6.1. All collected hardware input events from programming tasks, and what each hardware event represents

Hardware Event Name	Amount Collected	Hardware Device	Notes
Mouse Down	26,568	Mouse	Baseline hardware input. Occurs when any mouse button is pressed. The specific mouse button (left, middle, right) is determined through the "button" value in the event (0, 1, 2, respectively).
Mouse Up	26,466	Mouse	Baseline hardware input. Occurs when any mouse button is released. The specific mouse button (left, middle, right) is determined through the "button" value in the event (0, 1, 2, respectively).
Click	24,332	Mouse	Software interpretation of baseline hardware inputs. Occurs when there is a mouse down and mouse up event for mouse button 0 (left click). Does not occur for the middle or right mouse buttons.
Double Click	2,160	Mouse	Software interpretation of baseline hardware inputs. Occurs when two mouse clicks (mouse down, mouse up, mouse down, mouse up) occur within a short time frame. Does not occur for the middle or right mouse buttons.
Key Down	298,212	Keyboard	Baseline hardware input. Occurs when any key on the keyboard is pressed. If the key is held down, there will be an initial event when the key is first pressed, and then multiple events at regular intervals while the key is held down.
Key Up	268,998	Keyboard	Baseline hardware input. Occurs when any key on the keyboard is released.

Figure 6.1 details how often individual keys were pressed by participants during the survey. Predictably, letter keys are the most pressed keys, being ~48% of all keyboard inputs.

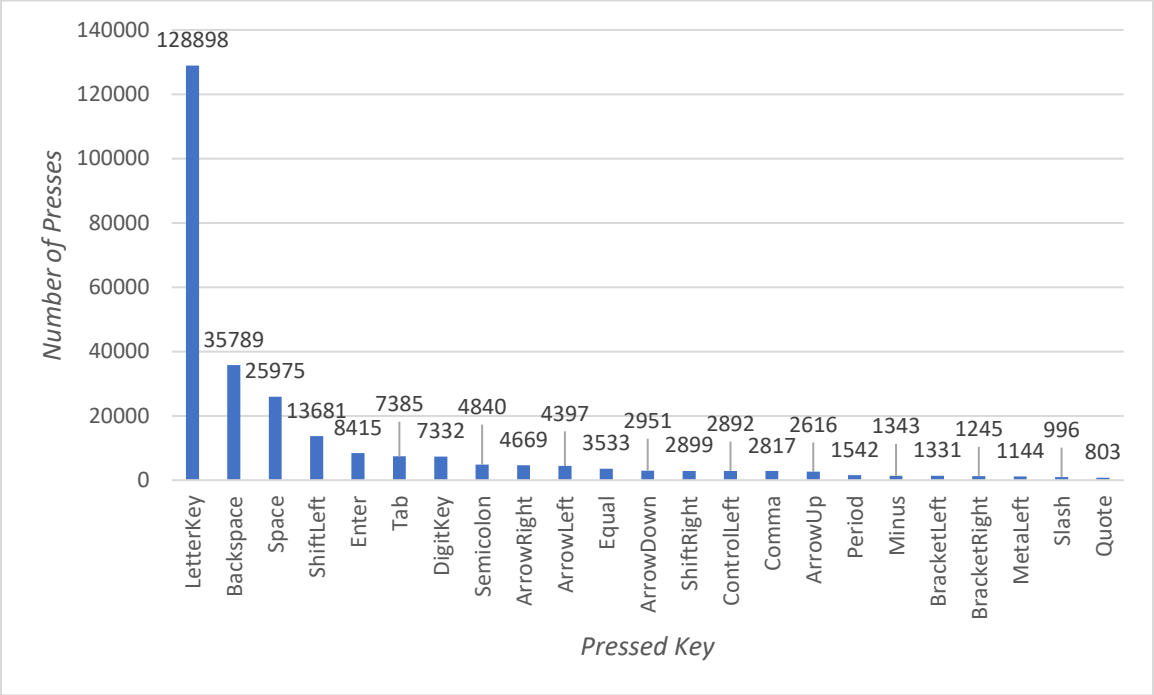


Figure 6.1. Collected keyboard keys pressed during programming tasks. The LetterKey column is the combination of all alphabetical keys on the keyboard, and the DigitKey column is the combination of all of the top-row digit keys on the keyboard. Note that this graph does not count the characters participants inserted into the text, only the physical keys pressed on the keyboard. For brevity, this chart only displays keys with over 500 key presses

6.1.2 Text Data

All changes to the source code made by participants during the survey are recorded. There are three types of text data collected: before input events, input events, and initial input events, as detailed in Table 6.2.

Table 6.2. All collected text data events during programming tasks, and what each event contains.

Text Data Event Name	Amount Collected	Notes
Before Input	234,165	Event triggered when a change is made to the text. Records the HTML input event type, the value inserted (if any), and the timestamp of the change.
Input	234,129	Event triggered when the text has been updated in response to a Before Input. Records the whole state of the text at the time of the event.
Initial Input	2,060	Event triggered when a task is loaded. Contains the text as initially shown to the participant, as well as a timestamp indicating when the specific task started. There is one initial input event per task.

Across all 2,060 programming task responses, 470,354 text data events are collected, representing 234,129 distinct textual changes to the target source code. All collected input types are identified through the W3C's standards of InputEvent attributes [W3C Editor's Draft 2025]. The input type is determined by the hardware inputs that cause the change and by what, if any, text is currently highlighted. For example, pressing the 'A' key triggers an insertText event, while inserting "A" via a clipboard paste triggers an `insertFromPaste` event. Because the survey was conducted via a simple HTML text area, not all HTML input events are available in the survey. Figure 6.2 details the total text event types collected during the survey.

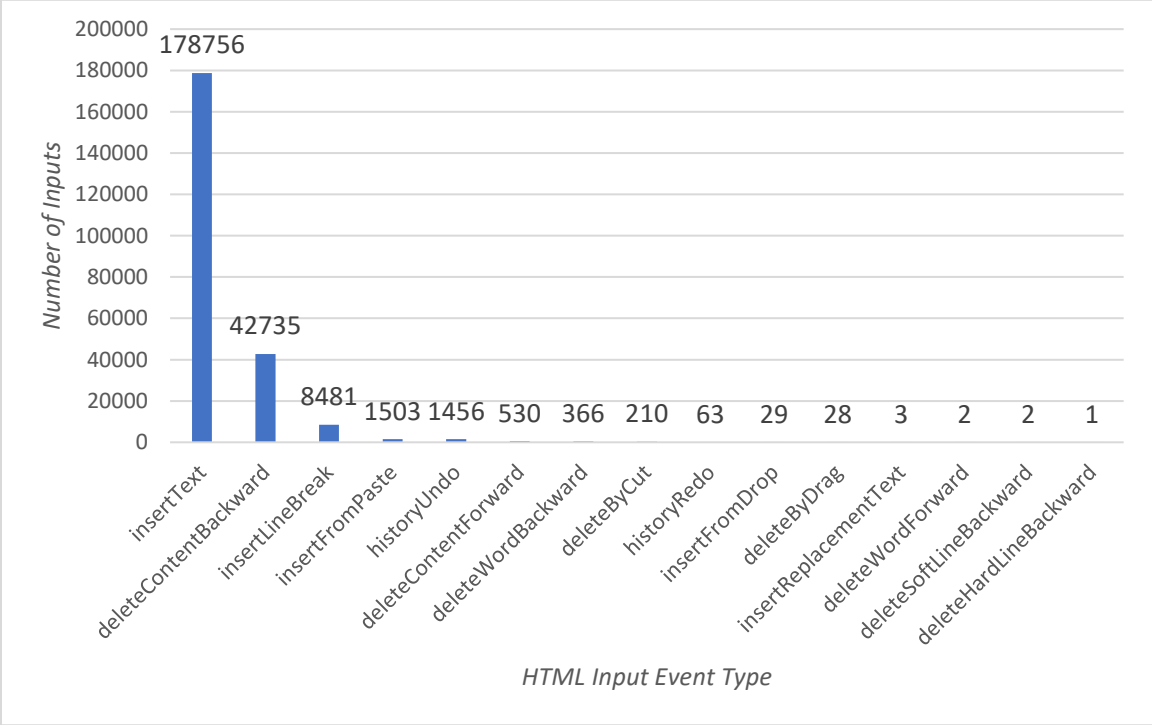


Figure 6.2. Input event types collected during programming tasks

It is important to note that one type of text change the survey tool did not correctly capture is tab-key indentation. In most development environments, pressing the tab key often inserts specific indentation for selected lines of code rather than replacing all selected code with a tab character. Most developers expect this interaction, but by default in a web browser, the tab key switches focus between elements on the web page. To make sure the editor felt closer to a real development environment, custom tab functionality was implemented into the survey tool to allow developers to indent and dedent. However, because this functionality is external to the text area, changes to indentation did not trigger any text events. Any changes to the indentation of the text persist and appear in the captured text event's code snapshot.

6.1.3 Selection Change Data

Lastly, all changes to the cursor/selection position are collected by the survey tool. The cursor position updates every time a change is made to the text area, and can also be adjusted manually with the mouse, arrow keys, or other hardware events. In total, 339,915 selection change events are collected from the 2,060 programming tasks.

6.2 Collected Prose Task Data

In total, 3.6 hours of editing are collected from the 235 participants across 230 prose tasks. Each participant spent an average of 57.6 seconds completing their prose task. The longest time a participant spent completing a prose task is 6.6 minutes, and the shortest is 17.9 seconds.

All information collected during the programming tasks is also collected during the prose tasks, meaning hardware input, text, and selection information is available for these tasks as well. In total, 80,960 hardware input events, 78,800 text data events, and 39,711 selection change events are collected from prose tasks. Table 6.3, Table 6.4, Figure 6.3, and Figure 6.4 detail the specific counts of event types during the prose tasks.

Table 6.3. All collected hardware input events from responses to the proseBaseline task

Hardware Event Name	Amount Collected
Mouse Down	529
Mouse Up	527
Click	295
Double Click	9
Key Down	39,805
Key Up	39,479

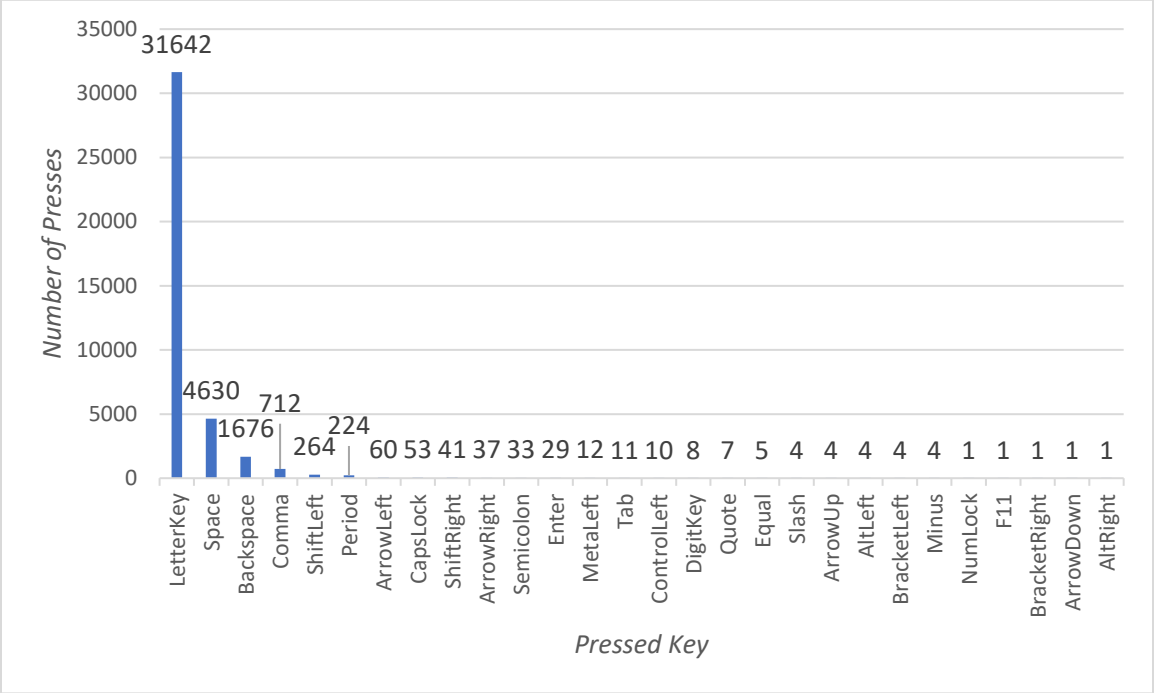


Figure 6.3. Collected keyboard keys pressed during responses to the proseBaseline task

Table 6.4. All collected text data events from responses to the proseBaseline task

Text Data Event Name	Amount Collected
Before Input	39,151
Input	39,106
Initial Input	230

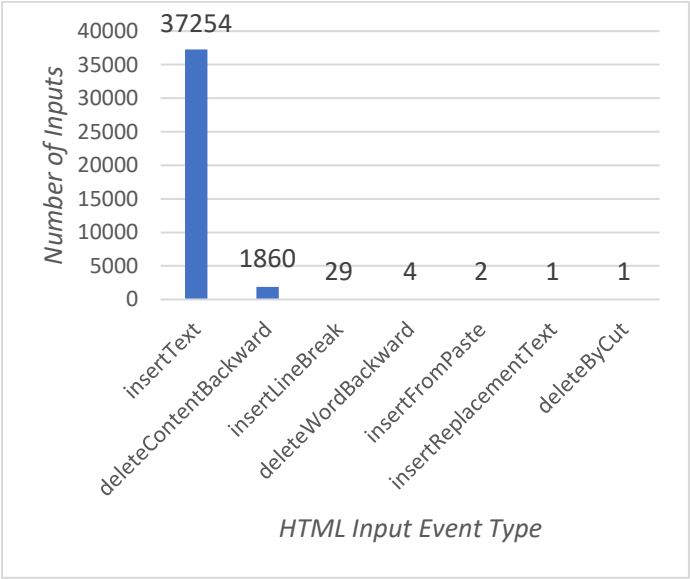


Figure 6.4. Input event types collected during responses to the proseBaseline task

6.3 Prose vs. Code Editing

Code and prose are not engaged with in the same way. Previous work finds that developers – especially experienced developers – read code far less linearly than prose [Busjahn, Bednarik, Begel, Crosby, Paterson, Schulte, Sharif, and Tamm 2015; Peitek, Siegmund, and Apel 2020]. While English prose is read left-to-right, up-to-down, code is read out of order, with many forward and backward jumps. This is, in part, due to the non-linear nature of code. While most programming languages execute top-to-bottom, they can rely on and call on things that are not necessarily in a predefined order or even available in the same file. Experienced developers are aware of this and are less likely to read code linearly, whereas novices are more likely to read it like prose.

To address RQ 1.2, this work explores some differences in typing and editing behavior of participants while answering the prose task and the programming tasks.

6.3.1 Cursor Positioning

When editing prose, developers are much less likely to adjust their cursor position manually. Of the 39,711 selection change events, 39,106 are caused by changes to the text from text events. This means that the remaining 605 changes are manual selection changes caused by the participant through interactions such as the arrow keys on the keyboard or mouse clicks. Conversely, out of the 339,915 selection events from the programming tasks, 105,786 are manual changes to the cursor position. Because there are ten programming tasks and only one prose task, the programming changes are normalized by a factor of ten, resulting in about 10,579 manual section changes per programming task. This is a difference of about two orders of magnitude compared to the prose task's manual selection changes.

6.3.2 Typing Speeds

Traditional typing speed tests measure typing speeds in words per minute (WPM). In this context, "word" does not refer to a space-separated series of characters, but instead means a sequence of five characters, including punctuation and whitespace [Arif and Stuerzlinger 2009]. Thus, WPM is actually a measure of the characters typed per minute divided by five and not an actual measure of individual words typed per minute. Typing speed tests also often record an accuracy score, as the goal of the typing test is to copy pre-provided text as fast and as accurately as possible. These metrics do not apply well to

source code, as it is rarely copied manually outside an academic or learning environment. In a work comparing novice computer science students' typing for source code and natural language tasks, Edwards et al. measure the speed of typing using digraphs – pairs of keys pressed one after the other [Edwards, Leinonen, Birthare, Zavgorodniaia, and Hellas 2020]. This speed, referred to as keystroke speed in this work, measures how fast individuals press keys on their keyboard.

This approach has the benefit of directly measuring how effective one is at typing without relying on the contents of the text of whatever they are editing. However, this does not provide a clear picture of an individual's skill in writing source code. A participant who is a fast typist but does not know any common keyboard shortcuts will be unable to complete certain tasks as fast as someone who is a slow typist but can make effective use of common editing shortcuts. Thus, another metric is needed to compare a developer's efficiency in editing source code. This metric, developed for this work, is known as character efficiency and measures how many characters have been added or inserted into the source code over the course of editing.

Another challenge in measuring typing speeds outside of typing tests is how to handle pauses in typing. When editing text, especially code, it is common for the editor to pause, whether to stop and think about what to do next or because they were distracted by an external stimulus. Edwards et al.'s work uses specific timing thresholds to determine whether a pause should be kept for typing speed purposes [Edwards, Leinonen, Birthare, Zavgorodniaia, and Hellas 2020]. For prose tasks, the pause threshold is 750ms, and for programming tasks, it is 2000ms.

Table 6.5 lists the calculated average character efficiencies and keystroke speeds across all task responses from the survey.

Table 6.5. Character efficiencies and keystroke speeds across all editing task responses. The "Programming Task Average" row is the average of all the above values. The character efficiency values are measured in the number of inserted/deleted characters per second, and the keystroke speed values are measured in the number of keystrokes per second

	Character Efficiency (chars/s)				*Keystroke Speed (keys/s)*			
Task	**Average**	**Median**	**Best**	**Worst**	**Average**	**Median**	**Best**	**Worst**
addParameter	5.26	3.69	252.48	1.74	3.43	3.27	6.50	1.58
commentCode	4.66	4.36	33.97	0.55	4.21	4.15	6.88	1.45
forToWhile_BubbleSort	4.44	3.32	110.51	1.03	2.68	2.57	4.82	1.46
forToWhile_ListSearch	3.45	2.98	24.89	1.45	2.74	2.56	10.30	1.33
functionsIntoClass	6.06	3.43	160.50	1.50	2.87	2.84	4.71	1.68
generalizeAvgFunctions	8.04	4.49	223.23	1.30	3.15	3.07	5.61	1.19
implementFileSummation	4.56	3.32	96.71	1.73	3.23	3.05	6.08	1.62
nameChange	4.00	3.58	36.65	1.13	3.42	3.39	8.69	0.95
removeDeadCode	6.89	3.61	84.09	0.75	3.01	2.86	7.98	0.75
renameFunction	4.62	3.87	44.26	1.86	4.32	3.91	16.70	1.85
Programming Tasks Average	5.20	3.67	106.73	1.30	3.31	3.17	7.83	1.39
proseBaseline	5.90	5.78	10.81	3.03	5.87	5.76	10.83	2.47

For the prose task, the keystroke speeds and the character efficiencies are roughly the same. This aligns with the task, as it simply involves copying the pre-provided text character by character. Participants are explicitly asked not to use the clipboard during this task, meaning almost every change made was a single-character change. Of the 39,151 input event types collected during the prose task, only 11 change more than one character at once.

The prose task invariably has faster keystroke speeds. This is consistent with the findings of Edwards et al., who also report faster keystroke speeds when participants wrote

natural language compared to source code. Interestingly, on average, the prose task also has a higher character efficiency, but when comparing the most efficient responses, the participants are more efficient when it comes to editing source code. For the majority of participants, even when performing programming tasks with clearer use cases for specific editing efficiencies like clipboard pasting, selection deletion, and more, participants are nonetheless less efficient writing code than they are writing prose. But, for an experienced programmer, they can reach far higher efficiencies editing code than they can writing prose, likely due to their application of such editing shortcuts.

It is important to note that the proseBaseline task involves participants copying a pre-existing sentence, while none of the programming tasks involve directly copying down pre-written source code. This difference could impact the speeds and efficiencies, giving prose editing a slight advantage. In future work, more kinds of prose tasks will be included to bridge this gap. However, one of the programming tasks can provide insight into writing prose that is novel, as opposed to copying pre-existing text.

6.3.3 Prose in Source Code

While source code as a medium is structured very differently than prose, prose can and often does exist within source code. Comments, and to a lesser extent, strings, often contain prose. Depending on their location and meaning, these comments and strings can contain simple phrases, whole sentences, or even paragraphs. While strings can contain prose, they are far less likely to due to the fact they have a semantic meaning within the source code and are often used in logic and processing. Comments, on the other hand, are used to document source code, and are typically sentence-like in structure. Because of this,

editing comments within source code can be evaluated as though it was someone editing prose. In the survey, the commentCode task involves participants reading a function and documenting it using a comment. Compared to the proseBasline task, the participants are allowed to use the clipboard if they desire and are writing new text, not copying pre-existing text.

Table 6.6. All collected text data events from responses to the commentCode task

Text Data Event Name	Amount Collected
Before Input	49,377
Input	49,382
Initial Input	214

In total, there are 52,652 selection change events in responses to the commentCode task. Because all text events in the source code will change the cursor's position, the 49,283 text events in Table 6.6 correlate to 49,283 selection events which are automatic changes to the cursor position. This leaves 3,369 selection events caused by manual cursor position changes. For the selection events of all the programming tasks, the number of previously calculated manual cursor changes needs to be recalculated without the commentCode task. Without the comment task, there are 287,263 selection events, 102,516 of which are manual. Normalizing this by a factor of nine for the nine non-comment programming tasks, there are about 11,391 manual selection changes per programming task.

Table 6.7. The number of manual selection events, both normalized and not, for the nine collective programming tasks, the commentCode task, and the proseBaseline task

Task(s)	Number of tasks in category	Number of manual selection events	Number of manual selection events normalized by number of tasks
Programming Tasks	9	102,516	11,391
commentCode	1	3,369	3,369
proseBaseline	1	605	605

Table 6.7 makes it clear that, while writing comments, developers manually change the cursor more often than they do when writing prose, but not nearly as often as they do when editing general source code.

For calculating average typing speeds, the commentCode task can also undergo the same analysis as the proseBaseline task, using the pause threshold of 750ms instead of 5000ms. When calculating the character efficiency and keystroke speed in this way, the results are similar to those for the prose task. Table 6.8 displays the speeds for both the poseBaseline and commentCode tasks. The only notable difference between the two is the best character efficiency speed, which is much higher for the commentCode task.

Table 6.8. Character efficiencies and keystroke speeds for the commentCode and proseBaseline tasks. In this table, the commentCode task is processed with the 750ms delay for prose tasks

	Character Efficiency (chars/s)				*Keystroke Speed (keys/s)*			
Task	**Average**	**Median**	**Best**	**Worst**	**Average**	**Median**	**Best**	**Worst**
commentCode	5.72	5.37	58.84	1.85	5.28	5.17	8.29	2.82
proseBaseline	5.90	5.78	10.81	3.03	5.87	5.76	10.83	2.47

6.4 Use of Editing Shortcuts

A part of writing source code is making use of specific computer interactions to edit text efficiently. As discussed previously, developers with knowledge of and skill using keyboard shortcuts can edit source code far more efficiently than someone relying on just typing one character at a time.

Two key shortcuts that allow for quick editing are utilizing the clipboard for copying and pasting text, and using the text cursor to highlight multiple characters and delete them simultaneously. Both of these shortcuts had clear use cases within the survey, with the generalizeAvgFunctions task being a clear application of copying and pasting, and the removeDeadCode task being a clear application of selection deletion. There are justifications for using these shortcuts throughout the entire survey as well.

Figure 6.5 and Figure 6.6 showcase the frequency of use of pasting and selection deletion, respectively. While slightly different, both graphs show a similar distribution, where many participants use the editing shortcut seldom or never, and a select few utilize the shortcut often. This is especially apparent for pasting, which 19% of participants never use.

There are many possible explanations for why a participant might not use a common editing shortcut, ranging from them not knowing about the existence of the shortcut to knowing about it, but believing that it goes against the study and that they should not use it. Future work is necessary to explore this seeming lack of computer and editor literacy, but this provides a partial explanation for why the average participant's character editing efficiency is lower for source code than prose.

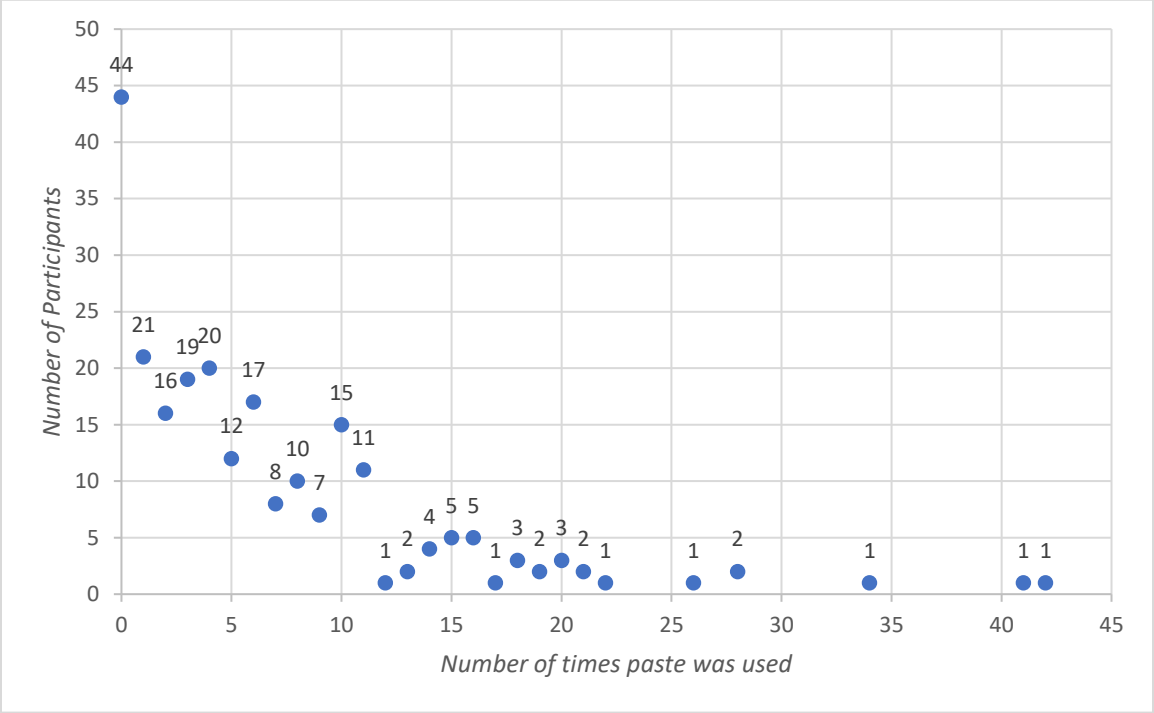


Figure 6.5. A scatterplot of how many participants pasted text during the programming tasks, and how often they did across the whole survey

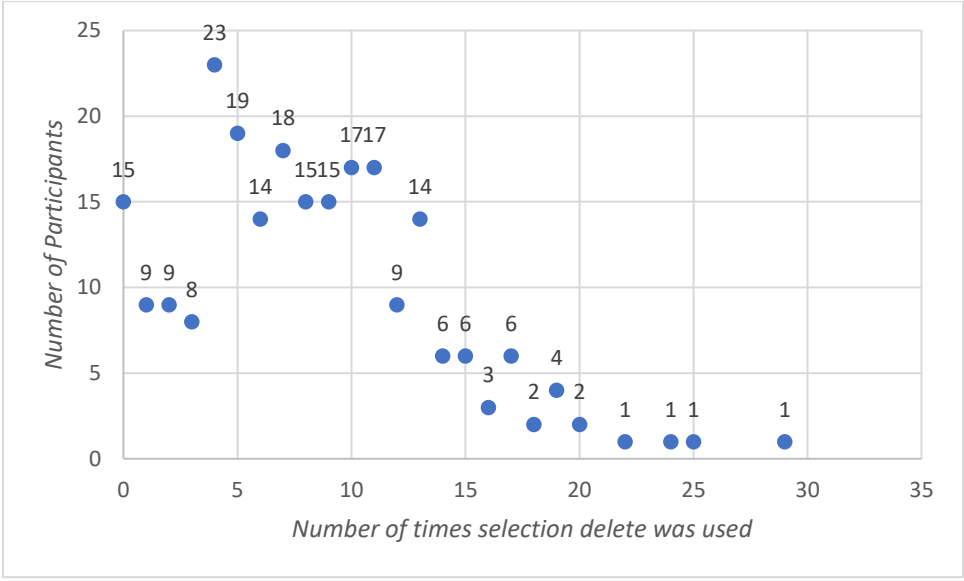


Figure 6.6. A scatterplot of how many participants selected and deleted text during the programming tasks, and how often they did across the whole survey

CHAPTER 7

Identifying Source Code Edits

To answer RQ2, this work investigates various methodologies for constructing source code edits from sequences of text events. As previously defined, a source code edit consists of a collection of related text events, each representing an atomic modification to the source code. However, it is not explicitly clear how text events should be grouped together and when they should be separated. Depending on the scenario or research needs, the criteria for creating source code edits can vary.

7.1 Creating a Reference Set of Source Code Edits

To better understand the properties of typical source code edits and address RQ2.1, a set of manually constructed and evaluated edits is required. These manually annotated edits, coded from participant judgement, serve as a reference dataset. This dataset provides both a baseline for evaluating algorithmic approaches and a foundation for determining heuristics to create such algorithmic approaches. It is important to note that this reference set is not meant to be used as a ground truth for identifying source code edits, but it is merely a first step towards defining useful approaches for identifying them.

7.1.1 Selecting Participants and Tasks

To create this reference set, six software engineering researchers are recruited as evaluators. Of the six, three are doctoral students (including the author) within the same lab and the other three are Ph.D. holding researchers, each at a different university. These six researchers are chosen for their expert experience in programming and their familiarity with software evolution research.

Next, the data that is to be evaluated needs determined. The evaluators look over a timeline of changes made to source code and grouping them into edits. Since the evaluators need authentic data of people making changes to source code, responses from the survey are ideal for this purpose. There are ten different programming tasks that are administered over the survey, each with varying lengths and requirements. To eliminate some bias from individual evaluators, each task response chosen will be examined by two of the evaluators and only edits independently defined by both will be put into the reference set. Choosing 30 task responses, three of each, satisfies the following properties:

- All evaluators examine one response from each task (ten total)
- Each response is examined by two evaluators
- Each unique pairing of evaluators occurs exactly twice

Using 30 task responses ensures every evaluator compares edits with every other evaluator, and every evaluator has roughly the same amount of work. To pick the 30 task responses from the survey dataset, a set of ten, one from each task, is selected randomly. Then, each response is either kept or rejected according to the following guidelines:

- For each type of task, there are responses from at least two different languages (C++, Python, Java)
- The response makes a reasonable attempt at completing the associated task prompt. It does not need to be a correct response, but it is a reasonable attempt
- The response is not overly long and is roughly around the average time for responses for that task. This ensures long responses where the participant paused for an extended period of time are not chosen.

7.1.2 Text Editing Replay Tool

To support the evaluators' understanding of textual changes made to the source code and make it simple to categorize them into edits, custom software is developed to display the task response. The tool, named the Text Editing Replay Tool, loads any response from the survey and replays it for the evaluators to review. Figure 7.1 shows the tool mid-replay of a task.

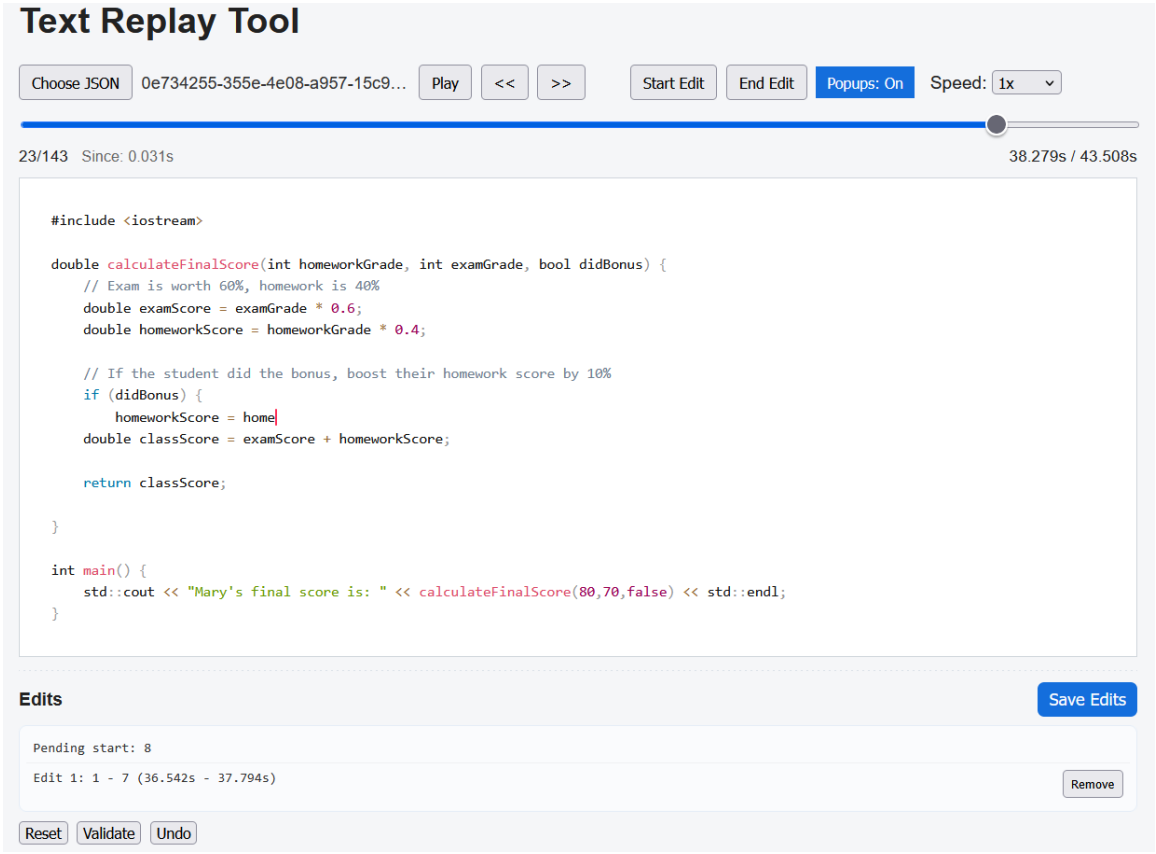


Figure 7.1. The Text Editing Replay Tool, mid replay of a response for the removeDeadCode task

Using the Start Edit and End Edit buttons (or the E hotkey) the evaluators can select the starting and ending text event for a specific edit. This is seen at the bottom of Figure 7.1 in the Edits section. The tool requires that every text event be grouped into an edit, a text event not be within multiple edits, and allows an edit to be defined with only a single text event. When all text events are grouped into edits, the edits can be saved into a data file for future analysis.

The Replay Tool provides many features for the evaluators to make their edit grouping easier. The tool can replay the entire session in real time, or it can be sped up or

down as desired. The three different types of editing events recorded during the survey are displayed in real time:

- Changes to the source code are updated as they happened, changing what text can be seen
- The cursor position updates to where it was for the participant during the survey. If any text was highlighted, the text also becomes highlighted during the replay
- Whenever a keyboard key or mouse button is pressed, a small popup above the text cursor appears, informing the evaluator of what the participant typed. These popups can be toggled on or off

The tool also allows stepping through just the text events, meaning an evaluator can skip through lengthy periods of downtime to examine just the text changes. If the evaluator makes a mistake with an edit, they can either undo a change or remove the edit entirely to correct the error. The Validate button allows an evaluator to check that they have processed all text events, and the tool will also refuse to save the edits if there are any text events not grouped into an edit.

7.1.3 Reference Set Results and Discussion

In total, the six participants defined 950 edits out of 3,625 text events. Since every task response has two evaluators grouping edits from its text events, the resulting edits from each evaluator can be compared. For the reference set, edits that are similar to each other, meaning the two evaluators agreed that the set of text events do comprise a single edit, are chosen. Edits that are exactly the same are called an exact match, and edits that

are off by one on either side (include an additional text event the other did not) are called a close match and are also counted. There are 426 edits that matched another (147 exact matches and 66 close matches), meaning the evaluators agreed on 44.8% of their edits.

After the six evaluators complete their assigned tasks, they meet to discuss their findings and opinions of what criteria are used to make an edit. All 213 agreed edits are compiled into a single HTML document, displaying the text events chosen, whether the match was exact or close, and how long the edit lasts. Additionally, a srcDiff [Decker, Collard, Volkert, and Maletic 2020] representation of the change is shown to make clear what the content of the edit is. Figure 7.2 showcases an example of what was shown during the meeting. This meeting was recorded, and various quotes from the meeting are used below to highlight the conclusions of the evaluators.

Task: addParameter_3 Sequence: 21-27 (exact match) People: (John,Josh) Time: 0.83s

Original

```cpp
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

Modified

```cpp
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders, bool ) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

Figure 7.2. An example of an edit agreed on by two of the evaluators and its visualization as shown during the evaluators' meeting to discuss the edits. In this example, the evaluators agreed that the participant adding a comma, space, and "bool" constituted a single edit

Task: addParameter_3 Sequence: 28-45 (exact match) People: (John,Josh) Time: 5.45s

Original

```
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders, bool) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

Modified

```
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders, bool noOrderRevenue) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

Figure 7.3. Another example of an agreed-on edit. This edit is the immediately following edit to the one depicted in Figure 7.2

One of the first things discussed was how timing affects the creation of edits. In Figure 7.2, the two evaluators agreed that the participant starting a new parameter by introducing a comma to separate it and adding the type of the parameter (`bool`) constituted a single edit. Figure 7.3 depicts the immediately following edit, which both evaluators also agreed on. When these edits were shown to the other four evaluators, who until this point had not seen this task, they expressed confusion as to why the two evaluators chose to separate them into two edits instead of just making them one. In this instance, there is a 46 second gap between "`, bool`" and "`noOrderRevenue`" being inserted, which is why the two evaluators decided to separate the two. One evaluator commented on this, saying "That makes sense, there is a big gap in time. I consider that two separate edits."

The next topic discussed was how changes to whitespace are factored into edits. Two evaluators noted that they had noticed a lot of instances where a participant inserts a number of new lines, pauses, deletes them, and then pauses again before making any non-whitespace changes. “If they do ‘enter enter enter’ and then ‘delete delete delete’ and pause, I do not consider that a part of any other edit.” It was also unclear how whitespace is associated with non-whitespace changes, if at all, with one evaluator saying, “I do not know whether the whitespace is included at the start of an edit, at the end, or separated entirely.” After some discussion, the consensus came to whitespace being grouped with whatever text events are closer, relying on timing information for grouping.

However, there is no clear answer to how the time between text events plays a role in edit making. One evaluator explicitly stated they initially struggled utilizing time in their decision making, saying “It was sort of hard to understand the time gaps.” Each evaluator had a different number in mind to use as an edit cutoff, ranging from 1.5 seconds to ten seconds. The evaluators did agree that cutting an edit off on a basis of time cannot rely on a static number, “Time is not a golden standard because what if someone is a slower typer?”

One area there was a clear agreement on is that edits are, at a maximum, line based. The largest agreed-on edit by the evaluators is a single 39 character-long assignment statement written continuously, shown in Figure 7.4. This edit was discussed extensively by the evaluators, who noted: “It was constant typing,” “They do some backspaces and they clear stuff up, but it is one continuous edit,” and “The amount of text they have – the syntactic structure did not factor into it.”

Task: forToWhile_BubbleSort_2 Sequence: 98-143 (exact match) People: (Collard,Maletic) Time: 40.80s

Original

```
1  def bubbleSort(list):
2      n = len(list)
3      i = 0
4      while i < n-1:
5          j = 0
6          while j < n-i-1:
7              if list[j] > list[j+1]:

8      for i in range(n-1):
9          swapped = False
10         for j in range(n - i - 1):
11             if list[j] > list[j + 1]:
12                 list[j], list[j + 1] = list[j + 1], list[j]
13                 swapped = True
14
15         # If no two elements were swapped, then break
16         if not swapped:
17             break
18
```

Modified

```
1  def bubbleSort(list):
2      n = len(list)
3      i = 0
4      while i < n-1:
5          j = 0
6          while j < n-i-1:
7              if list[j] > list[j+1]:
8                  list[j], list[j+1] = list[j+1], list[j]
9      for i in range(n-1):
10         swapped = False
11         for j in range(n - i - 1):
12             if list[j] > list[j + 1]:
13                 list[j], list[j + 1] = list[j + 1], list[j]
14                 swapped = True
15
16         # If no two elements were swapped, then break
17         if not swapped:
18             break
19
```

Figure 7.4. The longest agreed-on edit within the reference set. In this edit, the participant hand-copied an assignment statement from a lower part of the code

Additionally, the evaluators noted that comments did not seem to fit any of the criteria they mentally devised for edits in other parts of the source code. One evaluator said, "You know, it is not [source] text, it is prose. So, I think we need to look at it in a different way." Five of the six evaluators agreed that edits on comments should be broken up by words, while the other evaluator thought that dividing by sentences is a more appropriate method.

Outside of the criteria for creating edits, the evaluators also made comments about the behavior of some of the participants while typing. It was noticed that "They are worried about deleting any code, and so they make a copy of the code they are going to change, and then when they are done, they delete the original." One clear example of this is in the forToWhile_ListSearch_1 task (see APPENDIX B), where the participant made a copy of the function that needed updating instead of directly updating the given function. It was also noted by all of the evaluators that many participants chose to make manual copies of code instead of using a more efficient method such as utilizing the clipboard or highlighting

the text and dragging the text with the mouse while holding CTRL. The forToWhile_ListSearch_1 task also demonstrates this, and the example shown in Figure 7.4 shows the participant copying, almost character for character, a whole line manually. In response to this being pointed out for the example in Figure 7.4, one evaluator said, "Really that is a copy paste, just manual and one [character] at a time." These observations by the evaluators corroborate empirical data collected in Section 6.4. While this work does not explore the differences between novice and expert behavior due to a lack of expert responses, it was theorized by the evaluators that this recurring behavior is a symptom of the participants' general lower skill and knowledge level.

One evaluator summed up the general consensus on edit detection criteria, saying "There is a lot of agreement when we are changing formatting or whitespace. There is a lot of agreement when an entire line is added or removed. There is some agreement with regards to time delay, so if someone is typing in two or three tokens, and there is a long delay between a couple of them, then that affects the edit." All but one evaluator reported finding the tasks easier to complete over time and found them interesting to do. The other evaluator found the tasks challenging, saying "I found this to be difficult. I antagonized over every edit," and reported that it required a lot of thinking on their part to keep the criteria for segmenting edits consistent.

7.2 Edit Detection Algorithms

Relying on develops to identify edits from participant editing data is impossible. When creating the reference set, it took the six evaluators approximately nine man-hours to process 7,190 text events into 950 edits, or about 13.31 text events per minute. Therefore,

to process the entire survey data manually, it would take an estimated 17,584.10 man-minutes, or 392.10 man-hours. Automated approaches are needed to enable large-scale analysis.

There are many logical approaches for grouping text events into edits. The following work addresses RQ2.2, with subsections detailing various algorithmic approaches to this problem and a subsequent section comparing them and their performance against the manually created reference set. All the following algorithms operate on a single pass over the text events.

7.2.1 Simple Algorithm

The simple algorithm does not do any grouping of text events to form edits. Instead, it simply treats each individual text event as its own edit. This means that each edit represents an instantaneous change to the code instead of a change over time. This algorithm is analogous to how gazel [Fakhoury, Roy, Pines, Cleveland, Peterson, Arnaoudova, Sharif, and Maletic 2021] and CodeGRITS [Tang, An, Chen, Bansal, Huang, McMillan, and Li 2024] handle collecting changes to the source code for eye-tracking.

7.2.2 Temporal Gap Algorithm

A simple approach to grouping text events into edits is to separate them by time. Naturally, while working on source code, developers will work in bursts, typing quickly, with the only pauses typically being the fraction-of-a-second gaps between key presses, and then pause for longer when thinking or getting distracted. This starting and stopping is often not a long period of time; the average time between all text events recorded during

the survey is 1171.88ms. This average is also affected by outliers, when developers stop interacting with the task for long periods of time – sometimes hours. Removing the longest two gaps, both of which are over an hour long, brings the average gap to 884.03ms. Removing gaps longer than a minute (196 gaps) reduces the average gap to 774.97ms.

The time duration to use for edit segmentation is not easily chosen. There are many arguments for what a good cutoff point is. In Edwards et al.'s work, they use 750ms and 2000ms as a cutoff point for calculating keystroke speeds in prose and code contexts, respectively [Edwards, Leinonen, Birthare, Zavgorodniaia, and Hellas 2020]. When creating the reference set, the evaluators did not rely on a single, consistent time for dividing the edits, but agreed that 5000ms is a reasonable cutoff point for a typical edit. To test and compare the results of this algorithm when given different cutoff times, all three cutoff times (750ms, 2000ms, and 5000ms) are evaluated.

7.2.3 Dynamic Temporal Gap Algorithm

One issue with relying on a single cutoff point to determine edits is the varying level of typing ability participants. As seen in Table 6.5 and Table 6.8, the typing speeds of participants can vary wildly. If the gap cutoff time is set too small, the algorithm will create edits smaller edits from the text events of slower typists. Conversely, setting the cutoff time to be too large will make larger edits from faster typists.

To combat this, the temporal gap cutoff can be dynamically calculated for each participant. Instead of relying on a fixed time to cutoff edits, this algorithm uses the average gap between all text events multiplied by two as the gap cutoff. This ensures that the gap used to segment edits relates to how fast a typist the participant is. However, this algorithm

is affected heavily by long pauses. If a participant gets distracted and stops typing for an extended period of time, this pause can overinflate the average gap time, leading to a disproportionately big cutoff for edits.

7.2.4 Token Algorithm

Like words in prose text, source code is segmented into discrete chunks of characters: tokens. Each token is a series of alpha-numeric or symbol characters that make up some fundamental syntactic purpose within the source code. Because each token represents some syntactic purpose within the code, adding, deleting, or changing a token has some semantic purpose. This algorithm takes advantage of this fact by segmenting edits based on the token being changed within the text events.

Each text event is run through srcML to extract and separate the syntactic tokens within the source code. These tokens are stored sequentially within an array. Then, for each sequence of text events which contain a change to the same token, an edit is made. This also applies for text events that create new tokens or delete tokens. Additionally, any text event that affects more than one token at once, such as through pasting or highlighting and deleting, is counted as its own, separate edit.

7.2.5 Heuristics Algorithm

The heuristics algorithm is an edit detection algorithm that utilizes criteria derived from the evaluators who created the edits for the reference set. As discussed in Subsection 7.1.3, the evaluators came to a set of agreed conclusions about what criteria they believe should be used for grouping text events into edits. The criteria for grouping edits are:

- Some temporal cutoff is needed to segment edits
- Whitespace changes go with whatever edit grouping they are closest too, unless the temporal cutoff segments them to be by themselves
- Changes to the same line, so long as they are continuous, should be a single edit

The heuristic algorithm, as implemented for this work, is an extended version of the temporal algorithm with additional conditions (the criteria mentioned above) that end the currently collected edit and start a new one. Like the temporal algorithm, the heuristics algorithm is evaluated with static cutoff gap times of 750ms, 2000ms, and 5000ms.

7.2.6 Heuristics+Dynamic Temporal Algorithm

As the evaluators noted during their discussions, relying on a static maximum time for segmenting edits will not work equally well across all participants, as each participant will have a different level of skill when it comes to editing source code. The dynamic temporal algorithm is a way to dynamically adjust the cutoff times for these different editing efficiencies, and the heuristics+dynamic temporal algorithm is the same algorithm but with the additional cutoff conditions of the evaluator-defined heuristics.

7.3 Algorithm Evaluation

The above defined algorithms provide different approaches to calculating edits from sets of text events. It is important to note that this work does not claim that any of these approaches are the objectively best algorithm to use for creating edits. As mentioned previously, different approaches for creating edits will be better for different applications.

The goal of this work is to find a method of creating edits which approaches the idealized atomic edits – edits that maintain some semantic intent while still preserving the developer's direct changes to the source code. Atomic edits are important for software engineering studies that use eye-tracking, as researchers want to identify what the intentions of the developer are at any given moment while also maintaining an accurate snapshot of the current stimulus at a given time. Lacking a formal definition for atomic edits, the reference set serves as a baseline of what atomic edits are – semantically meaningful but concise changes to the source code, defined by expert software engineering evaluators.

In this section, the six algorithmic approaches are evaluated and compared against the reference set and each other. Various evaluation metrics are used to provide a general overview of what a given algorithm may excel at or fail at. The algorithms are then run on two datasets – first, the same task responses the evaluators used to create the reference set, and then the entire survey response dataset.

7.3.1 Evaluation Metrics

Edits are multidimensional data points, containing information related to both developer behavior, syntactic changes, and semantic meaning. To evaluate the algorithms, many metrics are needed so each aspect of an edit can be considered. Table 7.1 details these metrics.

Table 7.1. The various metrics used to evaluate the edit detection algorithms

Edit Algorithm Evaluation Metric	Metric Type	Definition
Number of Calculated Edits	Edit Structure	The total number of edits the algorithm creates from the input dataset of text events
Average/Median Text Events per Edit	Edit Structure	The average and the median number of text events per edit
Average/Median Duration	Edit Structure	The average and the median time duration of each edit in milliseconds
Distribution of Edit Category	Edit Structure	The distribution of edit categories of edits as generated by the algorithm (See Table 4.2)
Percentage of Syntactically Correct Edits	Source Code	The percentage of edits which result in syntactically correct code. The code may not be compliable or correct.
Exact and Close Matches with Reference Set	Reference Set	The number of exact and close matches of edits within the reference set when the algorithm is run on the same task responses.

To calculate each metric, the set of task responses is fed, one at a time, through the algorithm, with each response being stored in a table within a SQLite database. The schema of this table can be seen in Figure 8.2. After all responses are processed, SQL queries are used to capture the above metrics, except for the syntactically correct metric.

For the syntactically correct metric, the Tree-sitter tool is used to evaluate the syntax of the edit. Tree-sitter, among other things, is a parsing library for many different programming languages [Brunsfeld et al. 2026]. Tree-sitter's ability to report syntax errors in provided source code is vital for this metric. To calculate this metric, the source code from each edit is saved into a temporary file, and Tree-sitter is used to parse said temp file. If Tree-sitter reports any error after parsing, the edit is labeled as being syntactically invalid, while edits with no errors are reported as valid. srcML cannot be used for this purpose because, while srcML is designed to only be used on syntactically correct code, it will still parse incorrect code and create well-formed output without complaint. srcML

does not guarantee that its output will be correct or usable, but it will not crash, raise an error, or in any other way indicate that the provided code was syntactically incorrect.

There is an additional caveat to the syntactically correct metric. For all other metrics, all task responses are factored into the result regardless of the task or participant. However, in the case of the syntactically correct metric, any task response for the implementFileSummation_cpp task is ignored. This is because a semicolon was erroneously left out of the task's initial code state, meaning that unless a participant explicitly fixed the issue, every single text event, and by extension edit, is syntactically incorrect. This error can be seen in the C++ tasks in APPENDIX A. To avoid the bias this task would introduce, it is left out of calculations for the syntactically incorrect metric.

For the edit category metric, each edit's starting and ending text is processed through srcDiff[3] [Decker, Collard, Volkert, and Maletic 2020]. For each srcDiff output, all `diff` tags are extracted and tokenized. The tokens are then evaluated for whether they have been inserted, deleted, or replaced, and the edit category is determined from these token changes.

7.3.2 Algorithm Performance Compared to the Reference Set

To compare the algorithms against the reference set, the 30 task responses, originally given to the six evaluators to evaluate, are run through each algorithm and stored in a database.

[3] See https://srcDiff.org/

Table 7.2. The edit structure metric results for each algorithm and the reference set. Numbers that are bolded and underlined are the closest to the reference set's values. Numbers that are italicized and underlined are the farthest to the reference set's values

Algorithm	Number of Calculated Edits	Average Text Events/Edit	Median Text Events/Edit	Average Duration	Median Duration
Simple	3,595	1.00	1	0ms	0ms
Temporal 750ms	725	**4.96**	3	896.54ms	557ms
Temporal 2000ms	370	9.72	6	2,964.47ms	1,733ms
Temporal 5000ms	159	*22.61*	*14*	*10,938.06ms*	*6,180ms*
Dynamic Temporal	381	9.44	6	2,967.35ms	**1,600ms**
Token	1,221	2.94	1	873.67ms	0ms
Heuristics 750ms	775	4.64	3	816.57ms	460ms
Heuristics 2000ms	470	7.65	4.5	**2,154.84ms**	1,023ms
Heuristics 5000ms	339	10.60	6	4,124.07ms	1,988ms
Heuristics+ Dynamic Temporal	492	7.31	**5**	2,041.93ms	1,025ms
Reference Set	213[4]	5.30	5	2,531.08ms	1,370ms

Table 7.2 shows the results of each algorithm's results on the edit structure metrics. Besides the simple algorithm, which will always create the maximum possible number of edits, the token algorithm created the most edits and the temporal algorithm with a delay of 5000ms created the least. The number of edits created is not necessarily good or bad

[4] The reference set only contains edits that are agreed on by the evaluators, and therefore does not cover all text events over the 30 sessions.

either way, but the more edits an algorithm creates the less text events each edit will have, and vice versa. This is seen in the average/median number of text events per edit metric, with the 5000ms temporal algorithm having the most text events per edit in both, and the token algorithm having the least (ignoring the simple algorithm).

Interestingly, the algorithms with the closest average and median number of text events to the reference set are different. The 750ms temporal algorithm has the closest average, while the heuristics+dynamic temporal algorithm has an identical median. This implies that the heuristics+dynamic temporal algorithm groups edits together similarly to how the evaluators did for typical scenarios. In Figure 7.5, a visual comparison of the text events in edits is shown. The 750ms temporal algorithm has many smaller text event edits than the heuristics+dynamic temporal algorithm, which balances out the outliers for a closer average to the much more balanced reference set.

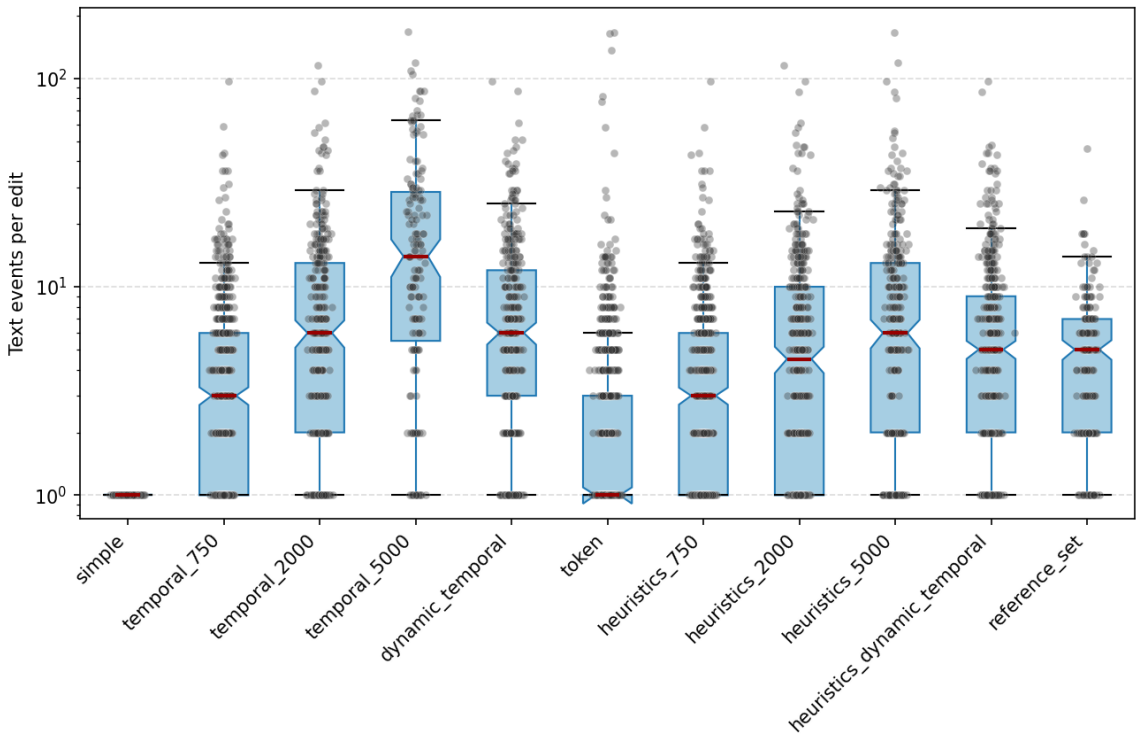


Figure 7.5. A logarithmic box and whisker plot of the number of text events per edit according to each algorithm and the reference set. The red bar in each represents the median

The duration metrics show a similar pattern. The dynamic temporal algorithm has the closest median to the reference set, while the 2000ms heuristics algorithm has a closer average. Like with the text events metric, this means that the dynamic temporal algorithm will create edits with durations similarly to the reference set for typical scenarios and the 2000ms heuristic algorithm has additional small duration edits to balance outliers, creating a closer average. This can be seen in Figure 7.6, with the dynamic temporal algorithm being the visually closest to the reference set, and the 2000ms heuristics algorithm containing many 0 duration edits.

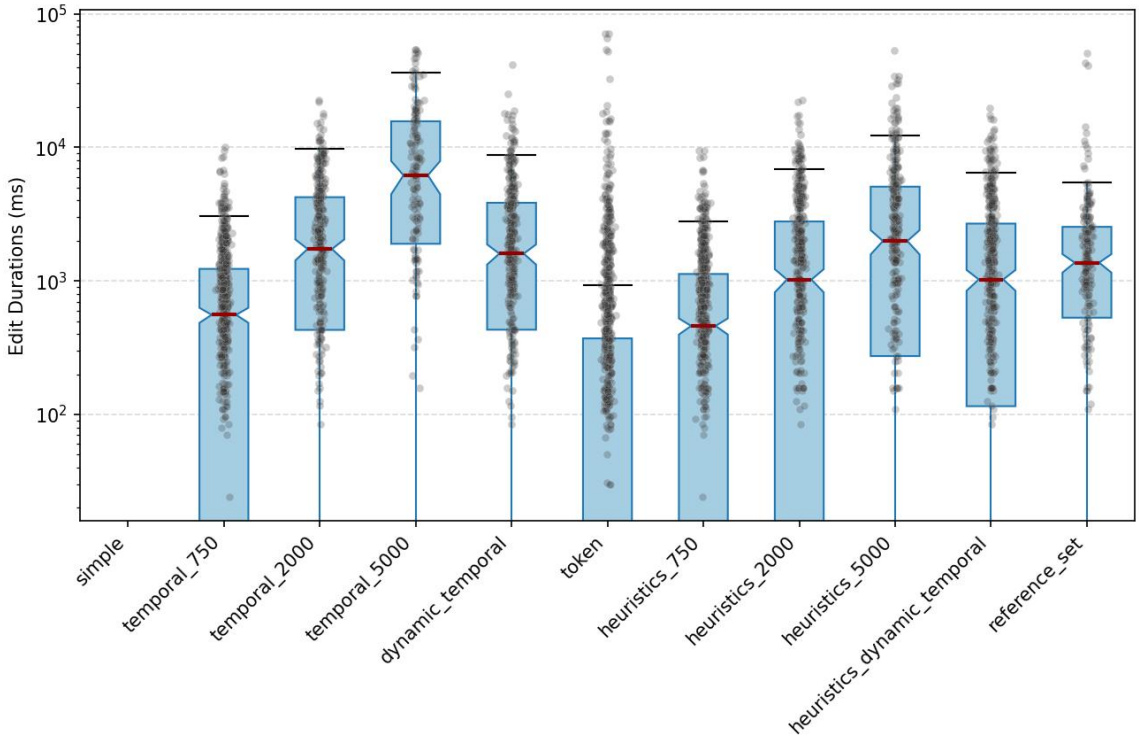


Figure 7.6. A logarithmic box and whisker plot of the durations of edits generated by each algorithm and the reference set. The red bar in each represents the median

Looking at both Figure 7.5 and Figure 7.6, it can be visually inferred that using a dynamic temporal gap matches the closest with the reference set. Additionally, in both figures, the 2000ms version of the heuristics algorithm and temporal algorithm, respectively, are very similar to their dynamic version, and for both metrics are the second-closest to the reference set's medians. This pattern shows that the dynamic temporal gap is calculated to be about 2000ms for the majority of participants, but that 2000ms is not effective for outlier participants, which the dynamic temporal algorithm can adapt to and match the reference set closely.

Table 7.3 details the evaluated categories of all the edits produced by the algorithms compared to the reference set. Note that, because edit categorization depends on srcML and srcDiff, there are certain situations where the tools fail to parse the before and after versions of the text from the edit. For example, certain syntactically invalid Python edits caused ill-formed XML output, leading to parsing errors. These cases are labeled as being not parsable.

There are clear differences in the kinds of edit categories each algorithm produces, with the shorter-edit algorithms producing more single-token edits than the longer-edit algorithms. This is intuitive, as the algorithms that produce shorted edits will have more edits, and these edits are likely to be comprised of single or few text events. The reference set leans has more multi-token edits than single-token edits, similar to the longer-edit algorithms.

Table 7.3. The edit categories generated by each algorithm on the reference set task responses and the reference set's edits

Algorithm	insertToken	insertTokens	deleteToken	deleteTokens	replaceToken	replaceTokens	wsFormatting	Not Parsable
Simple	530	39	128	49	1,793	202	848	6
Temporal 750ms	158	126	24	45	131	78	158	5
Temporal 2000ms	45	120	6	36	67	48	47	1
Temporal 5000ms	3	62	3	20	21	38	12	0
Dynamic Temporal	55	108	8	32	77	48	51	2
Token	403	49	85	48	138	158	334	6
Heuristics 750ms	165	124	24	46	136	80	195	5
Heuristics 2000ms	56	123	6	40	85	53	105	2
Heuristics 5000ms	16	106	4	33	64	44	71	1
Heuristics+ Dynamic Temporal	65	119	9	39	97	49	111	3
Reference Set	20	41	5	30	67	18	32	1

These structural metrics provide a general overview of the characteristics of the edits made by the algorithms, but it is important to note that they are operating on more data than the reference set. Since the reference set only contains the 213 edits agreed on by the participants, there are many text events, and even a whole task response, that are not represented in the reference set. This is the likely explanation for the more abundant outliers seen in the algorithms' outputs.

Table 7.4 details how well the algorithms produce edits that result in syntactically correct code. The reference set has the highest percentage, sitting at 73.21%. The best

performing algorithm is the temporal algorithm with a gap time of 5000ms. This is likely because the longest pauses a developer will take will be after finishing a statement or fixing something, which will result in syntactically correct code. This fact is supported by the fact that, of the heuristics algorithms, the 5000ms gap one also has the highest syntactic correctness. Conversely, the token algorithm generates the least amount of syntactically correct edits. This is likely because, when a new token is introduced, the majority of the time it will be for a new statement or construct, which will only be complete after a number of tokens are added. For example, to create a syntactically correct if statement, six tokens must be inserted:

- `if`
- An open parenthesis
- A close parenthesis
- Some condition, minimum one token
- An open curly brace
- A close curly brace

Additionally, in a language like Python, editing whitespace can cause the code to become syntactically incorrect.

Table 7.4. The syntactically correct metric results for the algorithms and the reference set. Entries that are bolded and underlined are the closest to the reference set's values. Entries that are italicized and underlined are the farthest to the reference set's values

Algorithm	Number of Edits	Number of Syntactically Correct Edits	Number of Syntactically Incorrect Edits
Simple	3,243	1,708 / 3,243 (52.67%)	1,535 / 3,243 (47.33%)
Temporal 750ms	660	366 / 660 (55.45%)	294 / 660 (44.55%)
Temporal 2000ms	342	202 / 342 (59.06%)	140 / 342 (40.94%)
Temporal 5000ms	151	**106 / 151 (70.20%)**	**45 / 151 (29.80%)**
Dynamic Temporal	351	214 / 351 (60.97%)	137 / 351 (39.03%)
Token	1,053	*378 / 1,053 (35.90%)*	*675 / 1,053 (64.10%)*
Heuristics 750ms	707	396 / 707 (56.01%)	311 / 707 (43.99%)
Heuristics 2000ms	437	266 / 437 (60.87%)	171 / 437 (39.13%)
Heuristics 5000ms	320	222 / 320 (69.38%)	98 / 320 (30.63%)
Heuristics+ Dynamic Temporal	456	287 / 456 (62.94%)	169 / 456 (37.06%)
Reference Set	209	153 / 209 (73.21%)	56 / 209 (26.79%)

The last metric to compare the algorithms is how many of the edits in the reference set they also made. To evaluate this, each of the edits in the reference set are compared against the entire list of edits generated by each algorithm. If there is an identical edit, it is counted as an exact match. If there is an edit that is close but is off by a text event in either direction, it is counted as a close match. Table 7.5 details the results of this process.

Table 7.5. How well each algorithm matched the edits within the reference set. Entries that are bolded and underlined are the largest values, and entries that are italicized and underlined are the smallest values

Algorithm	Exact Matches with Reference Set	Close Matches with Reference Set	Total Matches with Reference Set
Simple	*23 / 213 (10.80%)*	60 / 213 (28.17%)	83 / 213 (38.97%)
Temporal 750ms	110 / 213 (51.64%)	57 / 213 (26.76%)	**167 / 213 (78.40%)**
Temporal 2000ms	101 / 213 (47.42%)	31 / 213 (14.55%)	132 / 213 (61.97%)
Temporal 5000ms	36 / 213 (16.09%)	*12 / 213 (5.63%)*	*48 / 213 (22.54%)*
Dynamic Temporal	95 / 213 (44.60%)	35 / 213 (16.43%)	130 / 213 (61.03%)
Token	61 / 213 (28.64%)	**89 / 213 (41.78%)**	150 / 213 (70.42%)
Heuristics 750ms	104 / 213 (48.83%)	63 / 213 (29.58%)	**167 / 213 (78.40%)**
Heuristics 2000ms	117 / 213 (54.93%)	39 / 213 (18.31%)	156 / 213 (73.24%)
Heuristics 5000ms	93 / 213 (43.66%)	35 / 213 (16.43%)	128 / 213 (60.09%)
Heuristics+ Dynamic Temporal	**119 / 213 (55.87%)**	46 / 213 (21.60%)	165 / 213 (77.46%)

Overall, three algorithms performed well and matched over 75% of the edits in the reference set: the temporal algorithm at 750ms, the heuristics algorithm at 750ms, and the heuristics+dynamic temporal algorithm. The temporal and heuristic algorithms had very similar results – both achieving the highest percentage match at 78.40%, and the temporal algorithm having six more exact matches than the heuristic algorithm. The heuristic+dynamic temporal algorithm had two less overall matches but had ten more exact matches than the 750ms temporal algorithm. Conversely, the 5000ms temporal and simple algorithms are the worst at matching, with the former having the lowest total matches and

close matches, and the latter having the lowest exact matches. All other algorithmic approaches have above a 60% matching rate, with various exact and close match ratios.

While no algorithm is the best at all metrics, the heuristics+dynamtic temporal algorithm performs well across all metrics, being within the top four algorithms for all of them.

7.3.3 Algorithm Performance on Full Survey Dataset

Table 7.6 shows the metrics applied to each algorithm when ran on the entire survey dataset. In general, the results are consistent with the results from just the reference dataset. This can be seen visually in Figure 7.7 and Figure 7.8, which show box and whisker plots of the text event and duration metrics across both datasets. Across the two metrics, the algorithms produce similar results despite the massive dataset size increase from the reference data set to the entire survey data set. In Table 7.7, a similar distribution of multi-token and single-token edits can be seen, with larger-edit algorithms producing less single-token edits and vice versa.

These findings show that the task responses chosen for the reference set creation are a sufficiently representative set of responses from the greater survey dataset. Additionally, the algorithms have consistent output across many different participants with different typing habits and skills.

Table 7.6. The results of each metric across all algorithms when run on the entire survey dataset

Algorithm	Number of Edits	Average Text Events/Edit	Median Text Events/Edit	Average Duration	Median Duration	Percentage of Syntactically Correct Edits
Simple	233,144	1.00	1	0.00ms	0ms	46.57%
Temporal 750ms	40,545	5.75	3	1,011.00ms	537ms	48.63%
Temporal 2000ms	17,423	13.38	6	3,414.64ms	1,737ms	57.51%
Temporal 5000ms	8,756	26.63	13	11,376.14ms	6,180ms	63.07%
Dynamic Temporal	23,730	9.82	6	2,730.16ms	1,396.5ms	56.83%
Token	88,265	2.64	1	740.92ms	0.ms	30.63%
Heuristics 750ms	43,823	5.32	3	907.85ms	435ms	48.56%
Heuristics 2000ms	24,159	9.65	5	2,313.22ms	1,086ms	56.38%
Heuristics 5000ms	20,204	11.54	6	3,892.37ms	1,588ms	60.07%
Heuristics+ Dynamic Temporal	30,137	7.74	5	1,908.75ms	932ms	55.65%

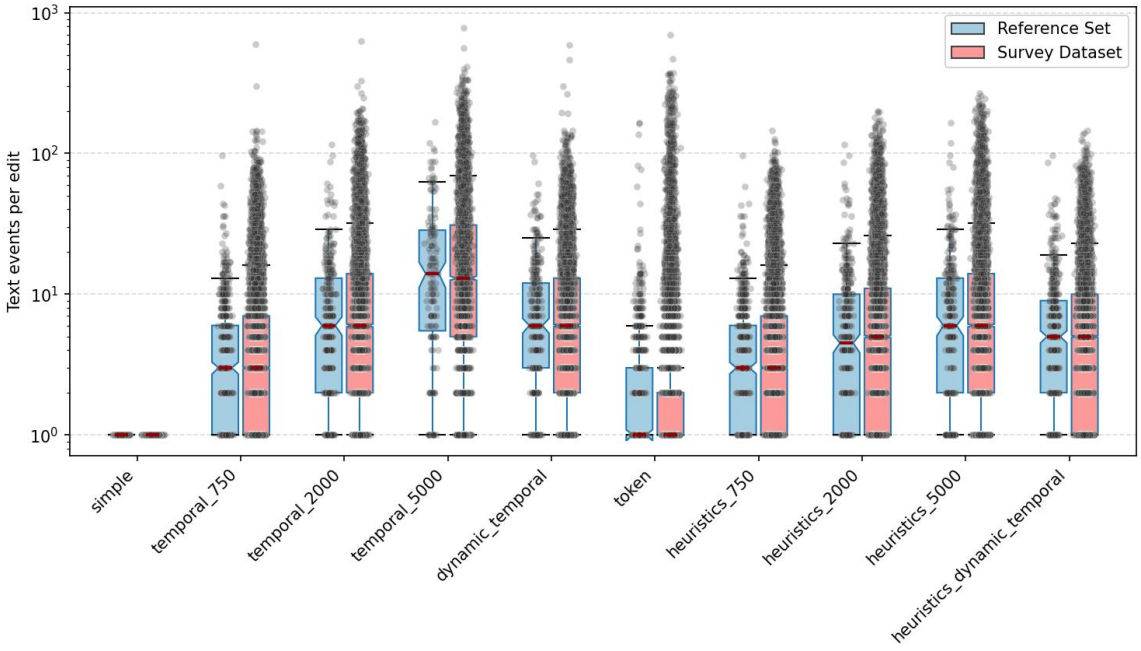


Figure 7.7. A logarithmic box and whisker plot of the number of text events per edit according to each algorithm across both the reference data set and the survey data set

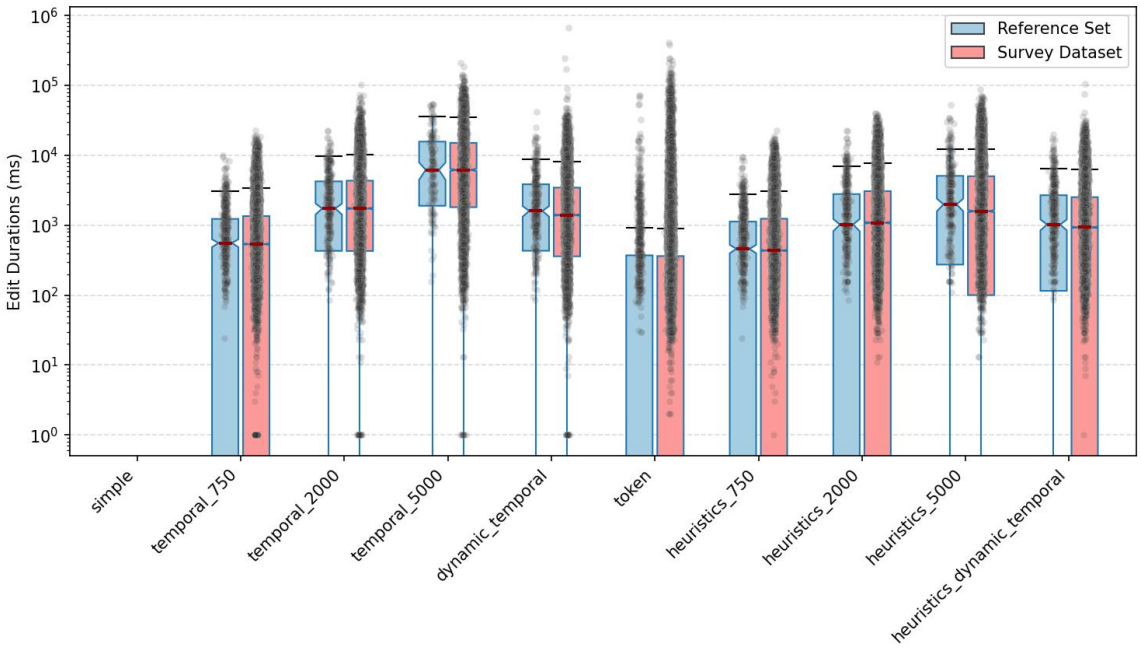


Figure 7.8. A logarithmic box and whisker plot of the number of the durations of edits generated according to each algorithm across both the reference data set and the survey data set

Table 7.7. The edit categories generated by each algorithm on the entire survey dataset

Algorithm	insertToken	insertTokens	deleteToken	deleteTokens	replaceToken	replaceTokens	wsFormatting	Not Parsable
Simple	33,932	2,558	8,988	2,469	123,914	15,222	45,621	440
Temporal 750ms	7,591	7.319	1,279	2,369	8,119	6,464	7,243	161
Temporal 2000ms	1,857	5,430	382	1,808	3,575	3,947	2,399	85
Temporal 5000ms	533	2,880	129	971	955	2,595	648	45
Dynamic Temporal	2,906	6,318	529	1,737	4,978	4,215	2,958	89
Token	23,304	3,278	6,248	2,373	9,626	12,025	21,158	253
Heuristics 750ms	7,804	7,406	1,316	2,452	8,235	6,619	9,821	170
Heuristics 2000ms	2,375	6,572	451	2,114	4,493	4,346	5,762	106
Heuristics 5000ms	1,167	5,807	250	1,935	3,213	3,513	4,246	73
Heuristics+ Dynamic Temporal	3,396	7,192	606	2,166	5,628	4,576	6,463	110

CHAPTER 8

Eye-Tracking with Editing Support

Traditional eye-tracking studies in software engineering typically rely on a static code stimulus, where the layout and document position of characters and tokens remain the same throughout the eye-tracking session. Editing introduces complications to this, as now any change can cause the relative document position of tokens to change wildly. Applying current token mapping techniques is thus insufficient for editing scenarios.

To address this gap and address RQ3, this chapter explores how eye-tracking studies can support editing scenarios through the application of previous findings in typing behavior and edit detection techniques.

8.1 Additions to the iTrace Infrastructure

To conduct eye-tracking studies that allow a participant to edit source code, new infrastructure is needed. The iTrace Infrastructure already has many tools developed for all stages of eye-tracking research on source code, meaning new functionality can be added to its existing tools instead of creating new ones.

The data collection and post-processing steps of iTrace's eye-tracking pipeline are given new functionality to support conducting eye-tracking studies with a code-editing component.

8.1.1 iTrace IDE Plugins

Arguably the most necessary part of support studies on developer editing behavior is to capture every minute change to the source code. As mentioned previously, gazel [Fakhoury, Roy, Pines, Cleveland, Peterson, Arnaoudova, Sharif, and Maletic 2021] and CodeGRITS [Tang, An, Chen, Bansal, Huang, McMillan, and Li 2024] are two previously existing tools that collect the changes made to source code over time. These approaches take a wholistic approach to capturing changes to code. Every time any change is made, the tool saves a copy of the entire source code file. This approach can lead to massive backlogs of data, especially if the eye-tracking participant is working on larger files. Because eye-trackers already produce vast quantities of data, it is ideal to reduce data collection where possible. To do this, instead of storing a snapshot of the file every time a change is made, the change itself is saved. This recorded change consists of what characters were inserted and/or deleted and the text position where this change occurred. While recording changes in this manner can also result in large records when a large body of code is pasted in or a large section of code is highlighted and deleted, for the majority of actual changes made to source code by developers, this will result in much less data. The downside to this approach is that the actual version of the text will need to be recreated at a later point during post-processing.

As initial work in updating the iTrace Infrastructure to support editing, this change collection is implemented into the iTrace-JetBrains plugin. iTrace-JetBrains is one of the newest plugins in the iTrace Infrastructure, and is notable for its ability to work on all of the different editors within the JetBrains family, such as IntelliJ, PyCharm and CLion.

iTrace-JetBrains is chosen as the initial plugin to be updates due to its ability to support studies in many different programming languages natively, and its relatively modern use compared to other editors supported by iTrace.

When an eye-tracking session is started, and iTrace-JetBrains is listening to iTrace-Core, it will immediately create event listeners for every single open document. These event listeners watch for any textual change in the code and, whenever something does change, it calls an asynchronous function that queues up a text event to be written into the plugin file alongside the IDE context data. A benefit to this document listening system is that it is independent of any hardware inputs, meaning that IDE tools such as Find/Replace, refactoring, and auto-formatting are captured. Figure 8.1 displays an example of how text events are saved within the iTrace plugin file.

```
<response event_id="1342446552229891 65" plugin_time="1779991922299" x="357"
y="437" gaze_target="forToWhile_BubbleSort.cpp" gaze_target_type="cpp"
source_file_path="C:/Users/srcdev/Desktop/Josh_Dissertation_Study/Participa
nts/John/forToWhile_BubbleSort.cpp" source_file_line="14"
source_file_col="6" editor_line_height="-1" editor_font_height="16.000000"
editor_line_base_x="" editor_line_base_y=""/>
<text_event timestamp="1779991922313"
source_file_path="C:/Users/srcdev/Desktop/Josh_Dissertation_Study/Participa
nts/John/forToWhile_BubbleSort.cpp" source_file_line="13"
source_file_col="5" inserted="i" deleted=""/>
<response event_id="1342446552230737 14" plugin_time="1779991922315" x="333"
y="427" gaze_target="forToWhile_BubbleSort.cpp" gaze_target_type="cpp"
source_file_path="C:/Users/srcdev/Desktop/Josh_Dissertation_Study/Participa
nts/John/forToWhile_BubbleSort.cpp" source_file_line="13"
source_file_col="4" editor_line_height="-1" editor_font_height="16.000000"
editor_line_base_x="" editor_line_base_y=""/>
```

Figure 8.1. An example of a text event being recorded in an iTrace plugin file next to IDE context responses. The text event is highlighted gray to make it stand out between the IDE context responses

8.1.2 iTrace-Toolkit

The next step to support editing studies with eye-tracking is to support the post-processing of eye-tracking and editing information. iTrace-Toolkit, iTrace's post-processing tool, is the natural fit for this. One of iTrace-Toolkit's main purposes is to map the line and column information collected from the IDE plugin to a specific source code token within the stimulus the participant was looking at. With editing involved, this process becomes more complicated, because now not that token mapping also needs to identify what version of the code existed at the point in time of the eye gaze and report the valid token from that.

To be able to accomplish this edit mapping, iTrace-Toolkit first needs to be able to load in the text event information from the plugin file. The iTrace Database needs updating to support the new types of information that will be stored in it. Four new tables are created for the database: `text_event`, `edit`, `edit_text_event`, and `edit_run`. Figure 8.2 shows these new tables in context with the original tables of the database.

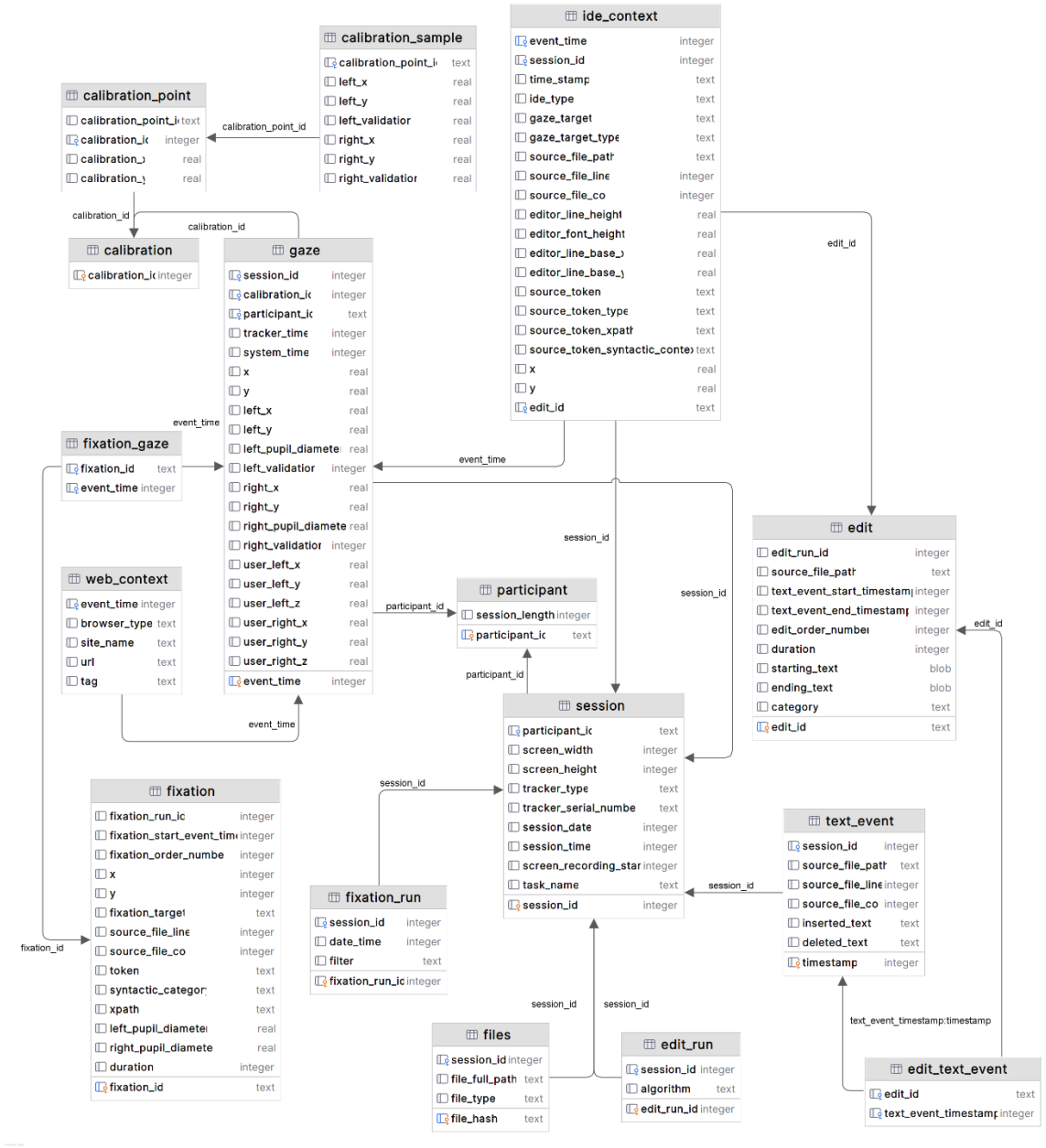


Figure 8.2. The Entity Relationship Diagram of the iTrace database file with editing information, updated from Figure 3.5

When loading eye-tracking sessions into iTrace-Toolkit, it will populate the `text_event` table with the text events from imported plugin files. These text events are then used later to calculated edits and map gaze data to specific tokens.

All six edit detection algorithms are implemented into iTrace-Toolkit, with the temporal and heuristic algorithms having separate options for the 750ms, 2000ms, and 5000ms time gaps. These edit algorithms are implemented into the token mapping menu, where a user can choose which edit algorithm to use when mapping tokens, or even to calculate edits at all. Figure 8.3 and Figure 8.4 show the GUI of iTrace-Toolkit where a user can choose what edit detection algorithm to use.

A srcML archive of the source code stimulus is needed by the token mapping process so that the line and column information can be mapped to a token within the source code. When calculating edits, a srcML archive of the original starting state of the stimulus is needed so the text events can be applied successively, and the edits can save the starting and ending versions of the source code. Previously, only a srcML file was needed by iTrace-Toolkit – however, with these changes iTrace-Toolkit now needs srcML to be installed on the same machine, as the tool now calls on srcML progressively during token mapping and when using the token algorithm.

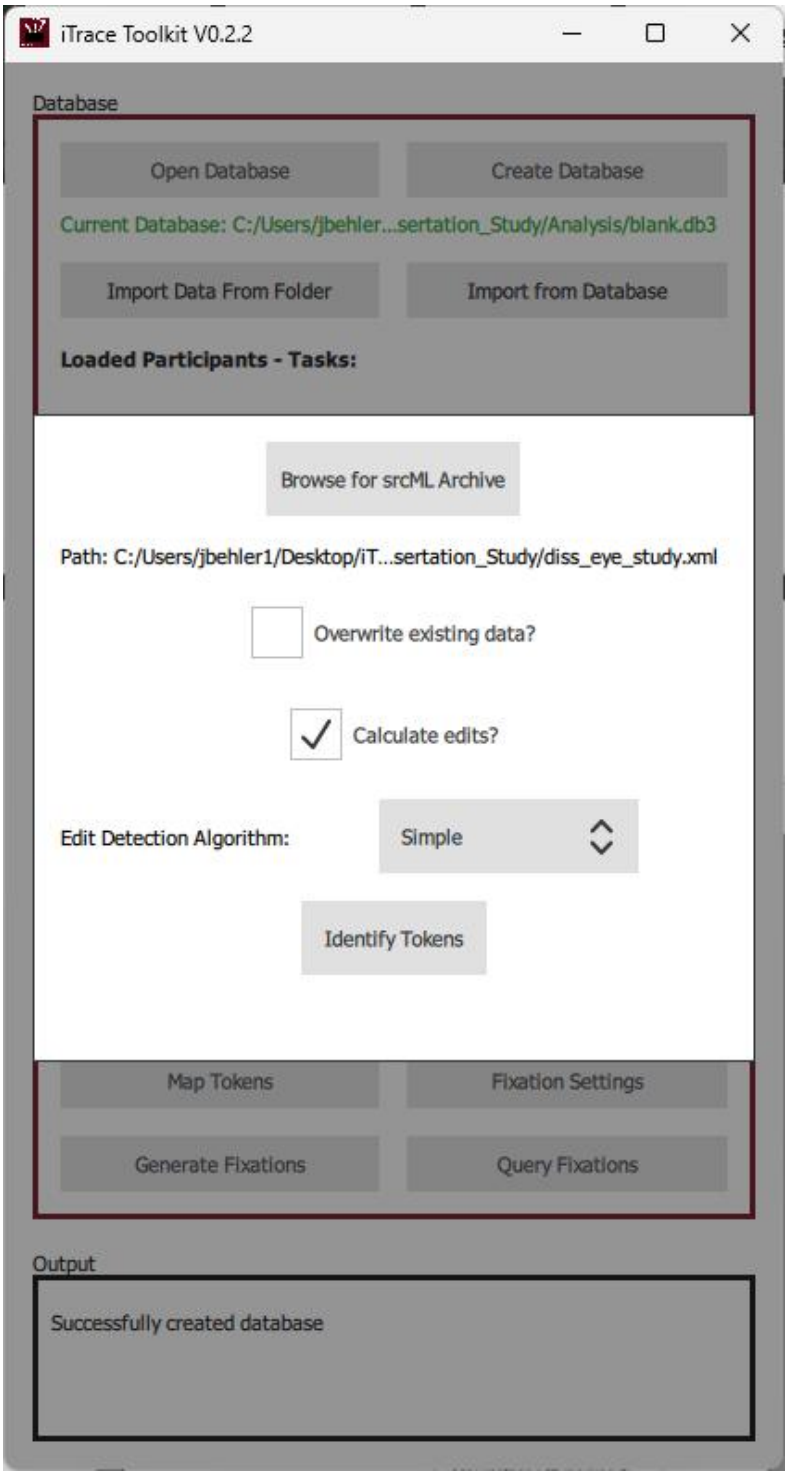


Figure 8.3. The token mapping menu in iTrace-Toolkit

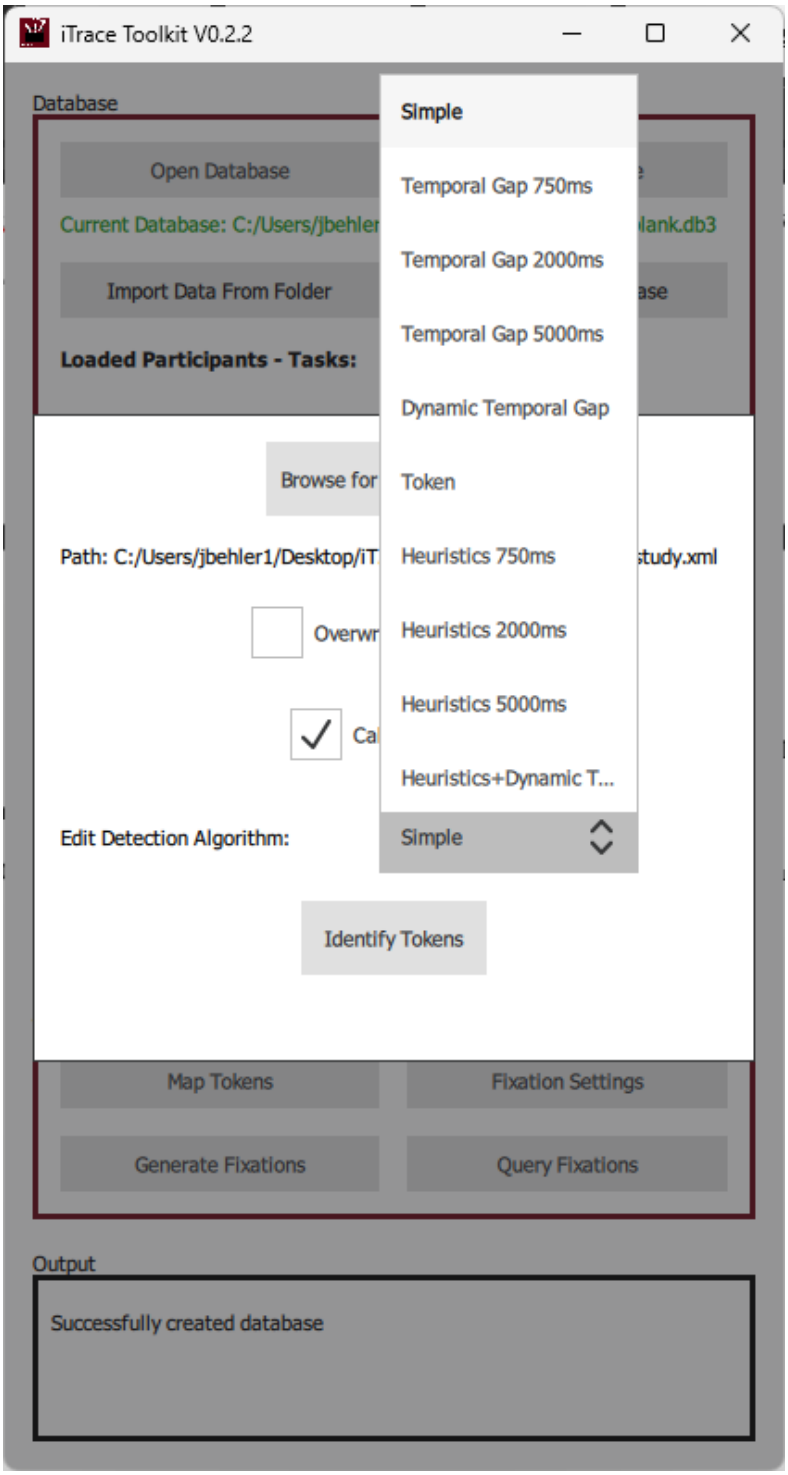


Figure 8.4. The edit detection algorithm drops down menu, listing the ten different editing algorithms that can be chosen

The ide_context table in the iTrace database is updated to include a new column, edit_id. This column is populated with the id of the edit that occurred alongside the gaze event, allowing for lookup of the text at that point. When edits are not calculated, a pseudo initial edit is created based on the srcML of the source code stimulus, allowing the text to still be gotten as needed during any post-processing.

8.2 Pilot Eye-Tracking Study Design

To test the efficacy of the functionality added to the iTrace Infrastructure, a pilot eye-tracking with code editing study is performed. Three programming tasks taken from the editing survey (forToWhile_BubbleSort, functionsIntoClass, and implementFileSummation) are given to participants. Importantly, the missing semicolon in the implementFileSummation task is fixed for this pilot study. Three participants are recruited for this task, all being graduate computer science students focusing on software engineering research. None of the recruited participants have previously completed the editing survey. Each task is loaded in the CLion development environment with iTrace-Jetbrains installed, and the participants are asked to edit the software in completion of the task. While editing, the participants' eye-movements are tracked using a Tobii Pro X3-120 eye-tracker recording at 120Hz. The participants recruited for this pilot study had not completed the editing survey, so had not seen the tasks before, and were given the option to complete the tasks in C++, Python, or Java; all participants chose to complete the survey in C++.

8.3 Pilot Eye-Tracking Study Results

Each participant completes three tasks, resulting in a total of nine eye-tracking sessions. In total, 17.72 minutes of eye-tracking and editing data is collected, made up of 127,631 gaze data points, 78,516 IDE context data points, and 902 text events. No problems are encountered during the eye-tracking sessions, with each session progressing identically to how a previous, non-editing eye-tracking session would. Table 8.1 contains the results of calculating all the algorithm evaluation metrics for every algorithm running on the data collected from the eye-tracking sessions.

Table 8.1. The results of each metric across all algorithms when run on collected eye-tracking study session data

Algorithm	Number of Edits	Average Text Events/Edit	Median Text Events/Edit	Average Duration	Median Duration	Percentage of Syntactically Correct Edits
Simple	902	1.00	1	0ms	0ms	257 / 902 (28.49%)
Temporal 750ms	148	6.09	4	1,015.61ms	452ms	79 / 148 (53.38%)
Temporal 2000ms	75	12.03	8	3,140.20ms	2,046ms	46 / 75 (61.33%)
Temporal 5000ms	33	27.33	21	10,874.91ms	7,071ms	23 / 33 (69.70%)
Dynamic Temporal	195	9.49	5	2,184.93ms	1,390ms	56 / 195 (58.95%)
Token	506	1.81	1	228.06ms	0ms	177 / 506 (34.98%)
Heuristics 750ms	172	5.24	3	815.68ms	381ms	97 / 172 (56.40%)
Heuristics 2000ms	121	7.45	4	1,648.84ms	909ms	81 / 121 (66.94%)
Heuristics 5000ms	101	8.93	4	2,504.62ms	1,073ms	73 / 101 (72.28%)
Heuristics+ Dynamic Temporal	134	6.73	4	1,352.96ms	629.5ms	85 / 134 (63.43%)

Because each individual gaze point matches with a specific edit, each edit detection algorithm will produce a unique set of gaze point-source code pairs, which shows what snapshot of the source code was used to map that particular gaze point to the code. Taking every unique set for each algorithm and finding the collective intersection of each, there are 30,512 common pairings out of the 69,890 non-initial gaze-point pairings. This means that, regardless of the algorithm used, roughly 43% of the gaze points are mapped to the same version of the source code for these nine eye-tracking sessions. This result is not completely unexpected, as regardless of what edit algorithm is used, the same text events are being used as inputs to all the algorithms, meaning that gaze points happening at specific points in time will invariantly have the same target source code snapshot, regardless of the specific text events that comprise it.

There are two types of gazes that are recorded during an eye-tracking study with code editing – gazes that occur while the participant is editing the text, and those that occur when the participant is not editing the text. Depending on the edit detection algorithm used to create the edits, a gaze could change from happening during an edit to happening between edits. Additionally, the screen location where a gaze could occur is also dependent on the edit, as it could be either on the code being changed, on code not being changed, or off the code entirely. Table 8.2 details how many gazes from the pilot study occur during an edit, and of those, how many occur on the area of code being changed, per algorithm.

Table 8.2. The number of gazes that occur during and not during an edit, and whether those gazes happen on code being edited, other code, or not on the code

Algorithm	Gazes During an Edit	Gazes Not During an Edit	Gazes on Changed Code	Gazes on not Changed Code	Gazes not on Code
Simple	9 (0.01%)	69,881 (99.99%)	0 (0.00%)	0 (0.00%)	9 (100.00%)
Temporal 750ms	14,780 (21.15%)	55,110 (78.85%)	3,256 (22.03%)	6,215 (42.05%)	5,309 (35.92%)
Temporal 2000ms	22,256 (31.84%)	47,634 (68.16%)	6,958 (31.26%)	7,772 (34.92%)	7,526 (33.82%)
Temporal 5000ms	35,668 (51.03%)	34,222 (48.97%)	17,512 (49.10%)	6,381 (17.89%)	11,775 (33.01%)
Dynamic Temporal	19,646 (28.11%)	50,244 (71.89%)	5,244 (26.69%)	7,751 (39.45%)	6,651 (33.85%)
Token	11,679 (16.71%)	58,211 (83.29%)	810 (6.94%)	6,055 (51.85%)	4,814 (41.22%)
Heuristics 750ms	14,055 (20.11%)	55,835 (79.89%)	3,026 (21.53%)	5,981 (42.55%)	5,048 (35.92%)
Heuristics 2000ms	19,256 (27.55%)	50,634 (72.45%)	4,483 (23.28%)	7,808 (40.55%)	6,966 (36.18%)
Heuristics 5000ms	24,895 (35.62%)	44,995 (64.38%)	6,668 (26.78%)	9,274 (37.25%)	8,953 (35.96%)
Heuristics+ Dynamic Temporal	17,618 (25.21%)	52,272 (74.79%)	4,258 (24.17%)	7,082 (40.20%)	6,278 (35.63%)

The algorithm that leads to the most gazes occurring during an edit is the 5000ms temporal algorithm, having more than 50% of its gazes during an edit. This is because this algorithm produces the longest edits, meaning more gazes that would otherwise fall between edits for other algorithms instead occur during the edit. Conversely, the simple algorithm produces the least, having only nine gazes occurring during an edit. This is due to the fact all edits with the simple algorithm have only text event, meaning a gaze must occur on the exact same millisecond as the edit for it to count as happening during the edit.

Using the heuristics+dynamic temporal algorithm as an example, there are 17,618 gazes that occurred during an edit. Of those gazes, roughly 25% occurred on tokens being edited by the participant, 40% occurred on other parts of the code, and 35% occurred with

the participant either looking away from any source code tokens or not looking at the screen.

Taking this analysis further, Table 8.3 details where each participant was looking while editing. Participant 2 spent very little time looking off-screen while typing. Participant 3 spent a large amount of time looking off-screen while typing – almost 20% of their time editing. This is consistent with observations made by the author during the eye-tracking session, where participant 3 reported being unfamiliar with the keyboard provided and was looking down at the keyboard often to make sure their hands were positioned correctly. Participant 1 was not seen consistently looking away from the screen while editing; however, participant 1 was wearing glasses, which can affect the eye-trackers accuracy. This also explains the higher proportion of gazes that appear outside of the code stimulus for participant 1.

Table 8.3. The number of gaze events and where they occurred per participant while editing

Participant	Gaze Events on Code	Gaze Events Elsewhere on Screen	Gaze Events Off Screen
Participant 1	4,007	4,884	743
Participant 2	3,521	1,586	34
Participant 3	4,169	1,629	1,233

Intuitively, each participant spent some time looking over the provided code before beginning to make any changes. Participants 2 and 3 spent a couple seconds at most reading through the source code before making their first change, while participant 1 spent the most time out of all participants reading the code before making their first change.

Table 8.4. Times spent before a participant started editing during a task

	Time before editing begins per task (ms)		
Participant	*forToWhile_BubbleSort*	*functionsIntoClass*	*implementFileSummation*
Participant 1	35,797	5,571	15,420
Participant 2	4,027	4,229	4,321
Participant 3	4,748	5,186	7,194

CHAPTER 9

Discussion

This dissertation investigates how developers interact with source code while editing it and how modifications to source code can be represented a format that preserves both semantic intent and developer actions. The findings presented throughout this work provide initial efforts towards supporting program comprehension research beyond static source code.

The implications of this work extend to multiple stakeholders. For researchers, the results highlight the importance of modeling developer behavior as a continuous and structured process of distinct semantic changes. For practitioners, particularly those involved in the design of development environments and developer tools, the code edits provide opportunities to align said tools with observed developer behaviors. For educators, the preliminary findings from the conducted editing survey highlight a potential need to reconsider how fundamental computer interaction skills are taught to prospective programmers early in their education.

9.1 Editing Behavior

Preliminary investigations into developer behavior during source code editing reveal key insights into how developers interact with their computing environment. From

the results collected in this work, it is shown that developers interact with the computer consistently and quickly while editing source code. Across all 2,060 collected editing task responses from the editing survey, there is an average of ~114 text events per response and ~314 hardware events per response. This, coupled with the average response time of 3.5 minutes per response, shows that developers perform distinct interactions with their computer in rapid succession while editing source code. These interactions vary, with the vast majority being single-key presses to insert or delete a character, but many others being more complex editing shortcuts that involve multiple hardware interactions to perform.

Despite the fact that participants are interacting with their computer often, collected evidence shows that developers do not edit source code efficiently. Across all metrics, the average participant is slower when editing source code compared to editing prose. This is compounded by the fact that there are a few highly efficient outliers who skew the average, meaning the average participant is much less efficient when the outliers are removed. Additionally, evidence suggests that participants, particularly novice participants, do not utilize editing shortcuts often, decreasing their potential efficiency while editing. This fact implies that novice developers either do not know about or are unaware of when to use editing shortcuts to increase how efficiently they can write source code. This has many implications for educators in computer science and could result in changes in how universities implement early computer science education, such as by introducing typing and editing education into courses.

9.2 Code Edits

Code edits model a level of software modification that has been almost non-existent in software engineering research. They allow for evaluation of both the semantic intent of the developer in terms of affecting the code base and the actions a developer takes to create such changes. This gives software engineering researchers further insight into the behavior of developers and allows for more effective models of software engineering and program comprehension. As shown through the pilot eye-tracking study, utilizing code edits allows empirical observations to be made on the behavior of developers while they are actively editing, and how that behavior affects their editing directly. Eye-tracking, while an obvious use, is not the only application of code edits in research and can also be used to explore questions such as how code smells, security vulnerabilities, and other interesting artifacts appear in source code, what differences a developer may have when editing pre-existing code versus writing new code, and previously mentioned applications for computer science education.

Additionally, code edits have use for industry practitioners as well. Through understanding and evaluation of code edits, the IDEs and tools that developers use on a daily basis can be improved to better suite typical workflow. Because many IDE features directly cause text events (Find/Replace, “Replace symbol under cursor”, etc.), understanding how these features are used in conjunction with overall development effort can allow these tools to be streamlined for typical workflows, further improving efficiency.

9.3 Threats to Validity

There are some threats to validity within this work that have been mitigated against but should be reported regardless.

The editing survey introduces some threats to validity. Because the survey is conducted over a tool ran in the web browser, it is not a standard environment for editing code, meaning that a developer might not edit code the same way they would in a code editor they are familiar with. Additionally, because of how the survey tool is implemented, certain behaviors may not work as intended. Undoing (CTRL+Z) functioned differently due to code indenting (Highlight+TAB) requiring custom implementation, meaning that if a participant indented their code, they could not undo past that change. This is mentioned to participants at the beginning of the survey, which could have altered how often participants used the undo feature. Additionally, due to an error in the survey tool, the action of indenting code was not properly collected as unique text event. The key press and cursor highlighting is collected, but the insertion of tabs is not collected until the next text event. Tab presses make up roughly 2.5% of all keypresses, meaning that these presses do not have a corresponding text event. However, the insertion of the tabs is still collected, just in addition to the actual next text event. This issue has been fixed in the survey tool, and any future studies conducted with the tool will not be affected by this problem.

The participants introduce another threat to validity. While the editing survey is open to any developer, the vast majority of participants are students currently in education.

Thus, the data collected by the survey may not be representative of overall developer behavior. Future work is needed to determine exactly how much editing behavior differs between novice and expert software developers.

There are some threats with the code edits that need to be addressed. The six evaluators for the code edit reference set are all software engineering researchers who have, to varying degrees, been involved or knew of this work as it was being explored. Thus, their opinions of what constitutes an edit may not be representative of what an average software engineering researcher could. Additionally, the six developed algorithms are all influenced by the heuristics defined by the six evaluators – especially the heuristics-based ones. Thus, both the generation and evaluation of code edits are based upon the heuristics and reference set defined by the evaluators. While this is mentioned previously, it is important to again note that this work does not posit that any of the proposed methods or approaches for calculating code edits are the best or the most optimal way to do so.

CHAPTER 10

Conclusions and Future Work

In this dissertation, the first steps towards understanding how developers edit source code are made. While prior work has largely focused on coarse-grained representations of code change, such as commits, revisions, and snapshots, this work instead investigates the finer-grained, moment-to-moment process through which source code is created and modified. Examining developer behavior at this level of interaction preserves the developer's moment-to-moment intent when writing the source code, contributing to a more complete and more realistic model of software development activity.

10.1 Results & Contributions

In total, eight research questions (three main questions, four sub-questions) are proposed and answered:

RQ 1. How do developers write and edit source code?

RQ 1.1. How do developers create changes to source code?

RQ 1.2. How does writing source code compare to writing prose?

RQ 2. What constitutes an atomic edit of source code?

RQ 2.1. What do developers consider an atomic edit?

RQ 2.2. Can atomic edits be programmatically identified?

RQ 3. How can eye-tracking studies map gaze data to a dynamically edited stimulus?

Broadly, these questions seek to (RQ1) characterize how developers interact with their computer to create and change source code, (RQ2) determine how sequences of these interactions can be meaningfully grouped into distinct edits of source code, and (RQ3) identify how eye-tracking infrastructure can be improved to support conducting eye-tracking studies with editing.

To answer these questions, a large-scale empirical study of developer editing behavior is done. A survey with ten different software editing tasks is undertaken by over two hundred participants, resulting in thousands of responses, each containing real-time editing information. These responses form a dataset that contains hardware input events, individual changes to source code text, and text cursor position changes, which provide a comprehensive representation of how the participants interacted with the code in completion of each task.

The analysis of this dataset reveals several findings regarding developer behavior and addresses RQ1. First, there are numerous ways developers can interact with source code. Even in the limited form of a survey within a web browser, developers can solve tasks in vastly different ways, and, depending on their skill level, can utilize different hardware interactions to accomplish these edits.

Second, editing source code differs substantially from writing prose. Prose is largely written sequentially, with little manual changes, while code is written non-linearly,

with the text cursor moving frequently. Comments, a prose-like construct found within source code, sits in-between normal prose and source code in terms of cursor movement.

Third, it is shown that survey respondents, despite the majority self-reporting as being of intermediate programming skill, write source code inefficiently – even less efficiently than writing prose. Respondents typically did not make use of various text editing shortcuts, such as Copy/Cut + Paste or Highlight+Delete. Even when accounting for natural gaps when working, participants are additionally slower at utilizing a keyboard when writing source code than writing prose.

To address RQ2, this dissertation explores how expert software engineers define singular, atomic edits of source code, and how these edits can be algorithmically calculated. Six software engineering researchers are tasked with manually labeling the starts and ends of edits from a sample set of survey responses. Their edits form a reference set of edits which are used to understand the criteria of a typical edit and form the basis for automatically detecting them. Using the reference set as a baseline, ten algorithmic approaches are evaluated and compared to each other. Various metrics related to the purpose and use of edits are used to evaluate them, with best-performing and worst-performing edits being noted.

Finally, to address RQ3, the iTrace Infrastructure is enhanced to support conducting eye-tracking studies that allow for participant editing of the source code stimulus. These changes, made within the iTrace-JetBrains IDE plugin and the iTrace-Toolkit post-processing tool, support the collection of and automatic processing of editing data alongside eye-tracking data. A small pilot study of three participants is conducted to

demonstrate the efficacy of these enhancements, and various findings from their collected data are noted to show that the approach has collected valid information.

10.2 Future Work

There are many opportunities for future work resulting from this dissertation. Firstly, the collected data set is mostly limited to university students. The dataset can easily be expanded to include more industry practitioners by distributing the study to public, online programming forums and outreach to local industry. Additional research into university level students to further understand their ability to efficiently edit source code and explore what gaps in their education can be filled to support them.

Secondly, the categorization of edits can be expanded. This work limited the categorization of edits to token-based categorization and did not assign semantic- or syntactic-level categories. Future work exploring how these edits can be categorized based on their semantic and/or syntactic effect on the source code, similar to the categories of micro-changes in previous work [Chen, Lanza, and Hayashi 2024], is planned.

Thirdly, there are many planned improvements to the iTrace Infrastructure to further support editing studies. iTrace has plugins for many IDEs and text editors, allowing researchers to conduct studies in whatever environment they think is best for their work. Until now, supporting many different editors was relatively simple, as all that is needed for an editor to work with iTrace is the ability to somehow map an x, y pixel screen coordinate to a specific line and column in the editor. That data is then written into an iTrace plugin data file, along with some editor environment information. However, supporting the collection of text changes introduces a new layer of complexity. Assuming an editor has

some API for allowing changes to the source code to be captured, there is no guarantee that its implementation will be the same, or even similar, to another editor's. For example, if a token is highlighted and then replaced with a single character, one editor may report that as a single replace action, while another may report it as a delete and then insert action. Additionally, one editor may report a copy+paste as a single insert action, while another may report each character insertion as separate insert actions. This means that, for identical actions in different editors, different text events might be recorded, which can affect how certain edit detection algorithms work. Future work is necessary to ensure that more plugins in the iTrace Infrastructure can capture editing information, and that all the plugins produce similar information as the others.

For the post-processing of eye-tracking data with editing information, work in this dissertation is limited to gaze point analysis. Future work will expand iTrace-Toolkit to support correctly mapping editing information to calculated fixations and saccades. Additionally, iTrace-Visualize currently does not support any visualizations with editing information. Extending the existing visualizations to support editing data and identifying new ways to visualize editing information is vital for researchers to be able to better understand the data they have collected.

CHAPTER 11

References

ALJEHANE, S., SHARIF, B., AND MALETIC, J. 2021. Determining Differences in Reading Behavior Between Experts and Novices by Investigating Eye Movement on Source Code Constructs During a Bug Fixing Task. *ACM Symposium on Eye Tracking Research and Applications*, Association for Computing Machinery, 1–6.

ANDERSSON, R., LARSSON, L., HOLMQVIST, K., STRIDH, M., AND NYSTRÖM, M. 2017. One algorithm to rule them all? An evaluation and discussion of ten eye movement event-detection algorithms. *Behavior Research Methods 49*, 2, 616–637.

ARIF, A.S. AND STUERZLINGER, W. 2009. Analysis of text entry performance metrics. *2009 IEEE Toronto International Conference Science and Technology for Humanity (TIC-STH)*, 100–105.

BEDNARIK, R., BUSJAHN, T., GIBALDI, A., ET AL. 2020. EMIP: The eye movements in programming dataset. *Science of Computer Programming 198*, 102520.

BEDNARIK, R. AND TUKIAINEN, M. 2006. An eye-tracking methodology for characterizing program comprehension processes. *Proceedings of the 2006 symposium on Eye tracking research & applications*, Association for Computing Machinery, 125–132.

BEHLER, J., CHIUDIONI, G., ELY, A., PANGONIS, J., SHARIF, B., AND MALETIC, J.I. 2023. iTrace-Visualize: Visualizing Eye-Tracking Data for Software Engineering Studies. *2023 IEEE Working Conference on Software Visualization (VISSOFT)*, 100–104.

BEHLER, J., WESTON, P., GUARNERA, D.T., SHARIF, B., AND MALETIC, J.I. 2023. iTrace-Toolkit: A Pipeline for Analyzing Eye-Tracking Data of Software Engineering Studies. *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 46–50.

BEHLER, J.A.C., VILLALOBOS, G., PANGONIS, J., SHARIF, B., AND MALETIC, J.I. 2024. Extending iTrace-Visualize to Support Token-based Heatmaps and Region of Interest Scarf Plots for Source Code. *2024 IEEE Working Conference on Software Visualization (VISSOFT)*, 139–143.

BINKLEY, D., DAVIS, M., LAWRIE, D., MALETIC, J.I., MORRELL, C., AND SHARIF, B. 2013. The impact of identifier style on effort and comprehension. *Empirical Software Engineering 18*, 2, 219–276.

BRITO, A., XAVIER, L., HORA, A., AND VALENTE, M.T. 2018. Why and how Java developers break APIs. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 255–265.

BROOKS, R. 1983. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies 18*, 6, 543–554.

BROWN, N.C.C., ALTADMRI, A., SENTANCE, S., AND KÖLLING, M. 2018. Blackbox, Five Years On: An Evaluation of a Large-scale Programming Data Collection Project.

Proceedings of the 2018 ACM Conference on International Computing Education Research, Association for Computing Machinery, 196–204.

BROWN, N.C.C., KÖLLING, M., MCCALL, D., AND UTTING, I. 2014. Blackbox: a large scale repository of novice programmers' activity. *Proceedings of the 45th ACM technical symposium on Computer science education*, Association for Computing Machinery, 223–228.

BRUNSFELD, M., QURESHI, A., HLYNSKYI, A., ET AL. 2026. tree-sitter/tree-sitter: v0.26.9. https://zenodo.org/records/20293153.

BUSJAHN, T., BEDNARIK, R., BEGEL, A., ET AL. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. *2015 IEEE 23rd International Conference on Program Comprehension*, 255–265.

CHANDRIKA, K.R. AND AMUDHA, J. 2025. Learner stimulus intent: a framework for eye tracking data collection and feature extraction in computer programming education. *Scientific Reports 15*, 1, 11860.

CHEN, L., LANZA, M., AND HAYASHI, S. 2024. Understanding Code Change with Micro-Changes. *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 363–374.

COLLARD, M.L., DECKER, M.J., AND MALETIC, J.I. 2011. Lightweight Transformation and Fact Extraction with the srcML Toolkit. *2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation*, 173–184.

COLLARD, M.L., DECKER, M.J., AND MALETIC, J.I. 2013. srcML: An Infrastructure for the Exploration, Analysis, and Manipulation of Source Code: A Tool Demonstration. *2013 IEEE International Conference on Software Maintenance*, 516–519.

DECKER, M.J., COLLARD, M.L., VOLKERT, L.G., AND MALETIC, J.I. 2020. srcDiff: A syntactic differencing approach to improve the understandability of deltas. *Journal of Software: Evolution and Process 32*, 4, e2226.

DIG, D., COMERTOGLU, C., MARINOV, D., AND JOHNSON, R. 2006. Automated Detection of Refactorings in Evolving Components. *ECOOP 2006 – Object-Oriented Programming*, Springer, 404–428.

DIG, D. AND JOHNSON, R. 2006. How do APIs evolve? A story of refactoring. *Journal of Software Maintenance and Evolution: Research and Practice 18*, 2, 83–107.

EDWARDS, J., LEINONEN, J., BIRTHARE, C., ZAVGORODNIAIA, A., AND HELLAS, A. 2020. Programming Versus Natural Language: On the Effect of Context on Typing in CS1. *Proceedings of the 2020 ACM Conference on International Computing Education Research*, Association for Computing Machinery, 204–215.

FAKHOURY, S., ROY, D., PINES, H., ET AL. 2021. gazel: Supporting Source Code Edits in Eye-Tracking Studies. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 69–72.

FALLERI, J.-R. AND MARTINEZ, M. 2024. Fine-grained, accurate and scalable source differencing. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, Association for Computing Machinery, 1–12.

FALLERI, J.-R., MORANDAT, F., BLANC, X., MARTINEZ, M., AND MONPERRUS, M. 2014. Fine-grained and accurate source code differencing. *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, Association for Computing Machinery, 313–324.

FLURI, B., WURSCH, M., PINZGER, M., AND GALL, H. 2007. Change Distilling:Tree Differencing for Fine-Grained Source Code Change Extraction. *IEEE Transactions on Software Engineering 33*, 11, 725–743.

GRABINGER, L., AL MADI, N., BEDNARIK, R., ET AL. 2025. A Cookbook for Eye Tracking in Software Engineering. *Proceedings of the 6th European Conference on Software Engineering Education*, Association for Computing Machinery, 60–76.

GRABINGER, L., HAUSER, F., WOLFF, C., AND MOTTOK, J. 2024. On Eye Tracking in Software Engineering. *SN Computer Science 5*, 6, 729.

GUARNERA, D.T., BEHLER, J.A.C., SHARIF, B., AND MALETIC, J.I. 2025. Automated Fixation Error Correction to Support Eye Tracking Studies on Source Code. *Proc. ACM Hum.-Comput. Interact. 9*, 3, ETRA04:1-ETRA04:17.

GUARNERA, D.T., BRYANT, C.A., MISHRA, A., MALETIC, J.I., AND SHARIF, B. 2018. iTrace: eye tracking infrastructure for development environments. *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, Association for Computing Machinery, 1–3.

HINDLE, A., GERMAN, D.M., AND HOLT, R. 2008. What do large commits tell us? a taxonomical study of large commits. *Proceedings of the 2008 international*

working conference on Mining software repositories, Association for Computing Machinery, 99–108.

KAGDI, H., COLLARD, M.L., AND MALETIC, J.I. 2007. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice 19*, 2, 77–131.

KOZAK, Z. 2026. A Webcam Eye Tracking Infrastructure for Software Engineering Tasks.

KULA, R.G., OUNI, A., GERMAN, D.M., AND INOUE, K. 2018. An empirical study on the impact of refactoring activities on evolving client-used APIs. *Information and Software Technology 93*, 186–199.

LEHMAN, M.M. 1980. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE 68*, 9, 1060–1076.

LEHMAN, M.M. 1996. Laws of software evolution revisited. *Software Process Technology*, Springer, 108–124.

LETOVSKY, S. 1987. Cognitive processes in program comprehension. *Journal of Systems and Software 7*, 4, 325–339.

LEVIN, S. AND YEHUDAI, A. 2017. The Co-evolution of Test Maintenance and Code Maintenance through the Lens of Fine-Grained Semantic Changes. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 35–46.

MARTINEZ, M., FALLERI, J.-R., AND MONPERRUS, M. 2023. Hyperparameter Optimization for AST Differencing. *IEEE Transactions on Software Engineering 49*, 10, 4814–4828.

VON MAYRHAUSER, A. AND VANS, A.M. 1997. Program understanding behavior during debugging of large scale software. *Papers presented at the seventh workshop on Empirical studies of programmers*, Association for Computing Machinery, 157–179.

MINELLI, R., MOCCI, A., AND LANZA, M. 2015. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. *2015 IEEE 23rd International Conference on Program Comprehension*, 25–35.

MYERS, E.W. 1986. AnO(ND) difference algorithm and its variations. *Algorithmica 1*, 1, 251–266.

OBAIDELLAH, U., AL HAEK, M., AND CHENG, P.C.-H. 2018. A Survey on the Usage of Eye-Tracking in Computer Programming. *ACM Computing Surveys (CSUR) 51*, 1, 5:1-5:58.

OLSSON, P. 2007. Real-time and Offline Filters for Eye Tracking. https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-106244.

PAN, K., KIM, S., AND WHITEHEAD, E.J. 2009. Toward an understanding of bug fix patterns. *Empirical Software Engineering 14*, 3, 286–315.

PEITEK, N., SIEGMUND, J., AND APEL, S. 2020. What Drives the Reading Order of Programmers? An Eye Tracking Study. *Proceedings of the 28th International*

Conference on Program Comprehension, Association for Computing Machinery, 342–353.

PENNINGTON, N. 1987a. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology 19*, 3, 295–341.

PENNINGTON, N. 1987b. Comprehension strategies in programming. In: *Empirical studies of programmers: second workshop*. Ablex Publishing Corp., USA, 100–113.

ROBBES, R. AND LANZA, M. 2007. A Change-based Approach to Software Evolution. *Electronic Notes in Theoretical Computer Science 166*, 93–109.

SALVUCCI, D.D. AND GOLDBERG, J.H. 2000. Identifying fixations and saccades in eye-tracking protocols. *Proceedings of the 2000 symposium on Eye tracking research & applications*, Association for Computing Machinery, 71–78.

SEGEDINAC, M., SAVIĆ, G., ZELJKOVIĆ, I., SLIVKA, J., AND KONJOVIĆ, Z. 2024. Assessing code readability in Python programming courses using eye-tracking. *Computer Applications in Engineering Education 32*, 1, e22685.

SHAFFER, T.R., WISE, J.L., WALTERS, B.M., MÜLLER, S.C., FALCONE, M., AND SHARIF, B. 2015. iTrace: enabling eye tracking on software artifacts within the IDE to support software engineering tasks. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, Association for Computing Machinery, 954–957.

SHARAFI, Z., SHAFFER, T., SHARIF, B., AND GUÉHÉNEUC, Y.-G. 2015. Eye-Tracking Metrics in Software Engineering. *2015 Asia-Pacific Software Engineering Conference (APSEC)*, 96–103.

SHARAFI, Z., SHARIF, B., GUÉHÉNEUC, Y.-G., BEGEL, A., BEDNARIK, R., AND CROSBY, M. 2020. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering 25*, 5, 3128–3174.

SHARAFI, Z., SOH, Z., AND GUÉHÉNEUC, Y.-G. 2015. A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology 67*, 79–107.

SHARIF, B. AND MALETIC, J.I. 2010. An Eye Tracking Study on camelCase and under_score Identifier Styles. *2010 IEEE 18th International Conference on Program Comprehension*, 196–205.

SHARIF, B. AND MALETIC, J.I. 2016. iTrace: Overcoming the Limitations of Short Code Examples in Eye Tracking Experiments. *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 647–647.

SILVA, D. AND VALENTE, M.T. 2017. RefDiff: detecting refactorings in version histories. *Proceedings of the 14th International Conference on Mining Software Repositories*, IEEE Press, 269–279.

SOLOWAY, E. AND EHRLICH, K. 1984. Empirical Studies of Programming Knowledge. *IEEE Transactions on Software Engineering SE-10*, 5, 595–609.

TANG, N., AN, J., CHEN, M., ET AL. 2024. CodeGRITS: A Research Toolkit for Developer Behavior and Eye Tracking in IDE. *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, Association for Computing Machinery, 119–123.

TAO, Y., DANG, Y., XIE, T., ZHANG, D., AND KIM, S. 2012. How do software engineers understand code changes? an exploratory study in industry. *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, Association for Computing Machinery, 1–11.

TSANTALIS, N., GUANA, V., STROULIA, E., AND HINDLE, A. 2013. A multidimensional empirical study on refactoring activity. *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, IBM Corp., 132–146.

TSANTALIS, N., KETKAR, A., AND DIG, D. 2022. RefactoringMiner 2.0. *IEEE Transactions on Software Engineering 48*, 03, 930–950.

TSANTALIS, N., MANSOURI, M., ESHKEVARI, L.M., MAZINANIAN, D., AND DIG, D. 2018. Accurate and efficient refactoring detection in commit history. *Proceedings of the 40th International Conference on Software Engineering*, Association for Computing Machinery, 483–494.

TURNER, R., FALCONE, M., SHARIF, B., AND LAZAR, A. 2014. An eye-tracking study assessing the comprehension of c++ and Python source code. *Proceedings of the Symposium on Eye Tracking Research and Applications*, Association for Computing Machinery, 231–234.

UWANO, H., NAKAMURA, M., MONDEN, A., AND MATSUMOTO, K. 2006. Analyzing individual performance of source code review using reviewers' eye movement. *Proceedings of the 2006 symposium on Eye tracking research & applications*, Association for Computing Machinery, 133–140.

VON MAYRHAUSER, A. AND VANS, A.M. 1995. Program comprehension during software maintenance and evolution. *Computer 28*, 8, 44–55.

W3C EDITOR'S DRAFT. 2025. Input Events Level 2. https://w3c.github.io/input-events/#interface-InputEvent-Attributes.

WALTERS, B., FALCONE, M., SHIBBLE, A., AND SHARIF, B. 2013. Towards an eye-tracking enabled IDE for software traceability tasks. *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 51–54.

XIA, X., BAO, L., LO, D., XING, Z., HASSAN, A.E., AND LI, S. 2018. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering 44*, 10, 951–976.

YUSUF, S., KAGDI, H., AND MALETIC, J.I. 2007. Assessing the Comprehension of UML Class Diagrams via Eye Tracking. *15th IEEE International Conference on Program Comprehension (ICPC '07)*, 113–122.

ZYRIANOV, V., PETERSON, C.S., GUARNERA, D.T., ET AL. 2022. Deja Vu: semantics-aware recording and replay of high-speed eye tracking and interaction data to support cognitive studies of software engineering tasks—methodology and analyses. *Empirical Software Engineering 27*, 7, 168.

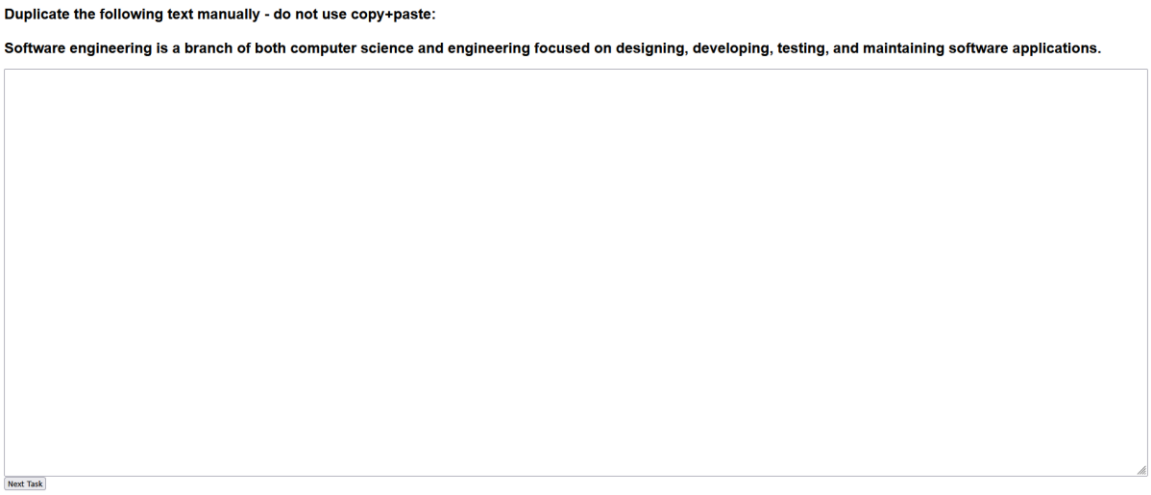
APPENDIX A

Survey Editing Tasks

The following is a complete list of all the editing tasks asked to participants during the editing survey.

Prose Task:

Task ID: proseBaseline



C++ Tasks:

Task ID: nameChange_cpp

Change all the variables named `x` to be `width` and the variables named `y` to be `height`.

```
#include <iostream>

int calculateArea(int x, int y) {
    return x * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

Next Task

Task ID: removeDeadCode_cpp

The function `calculateFinalScore` calculates a student's score in a course. The course is removing the bonus assignment, and the function needs to be updated. Remove all code that involves computing the bonus.

```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade, bool didBonus) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;

    // If the student did the bonus, boost their homework score by 10%
    if (didBonus) {
        homeworkScore = homeworkScore * 1.1;
    }

    double classScore = examScore + homeworkScore;

    return classScore;

}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

Next Task

Task ID: addParameter_cpp

The `calculateDailyRevenue` function calculates the per-day revenue gained by a delivery company. The function is going to be expanded to allow the optional calculation of potential revenue if no orders were cancelled. Add a new boolean parameter to the function that allows for this upgrade to be made.

```
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

Next Task

Task ID: renameFunction_cpp

Rename the function `foo` to have a more appropriate name.

```
#include <iostream>
#include <vector>

bool foo(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

Next Task

Task ID: commentCode_cpp

Write a comment above the `isStrongPassword` function that explains what it does along with the criteria for a strong password.

```cpp
#include <string>

bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

Next Task

Task ID: forToWhile_ListSearch_cpp

Convert the for loop in the following code to be a while loop.

```cpp
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Next Task

Task ID: functionsIntoClass_cpp

Create a new class called `Log` and encapsulate the following 3 functions into it as its methods. Make the `log` parameter a data member of the class and remove it from the parameter lists of the methods.

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

#endif
```

Next Task

Task ID: forToWhile_BubbleSort_cpp

Convert the for loops in the following code to be while loops.

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Next Task

Task ID: generalizeAvgFunctions_cpp

The following code contains two functions that average students' exam and homework scores. Replace the two specific functions with one general function that can work on both the exam and homework arrays. Move the printing to be in the main function.

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
void printHomeworkAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

Next Task

Task ID: implementFileSummation_cpp

Implement the `readAndSumNumbers` function. It should read two numbers from the provided file, print the sum of the two, then return the value.

```cpp
#include <iostream>
#include <fstream>

int readAndSumNumbers(std::ifstream& file) {

}

int main() {
    std::ifstream numbers_file("data.txt");
    if (!numbers_file.is_open()) {
        exit(1);
    }

    while(!numbers_file.eof()) {
        readAndSumNumbers(numbers_file);
    }

    numbers_file.close()
}
```

Next Task

Java Tasks:

Task ID: nameChange_java

Change all the variables named `x` to be `width` and the variables named `y` to be `height`.

```
import java.util.Scanner;

public class Main {
    static int calculateArea(int x, int y) {
        return x * y;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("What is the width? ");
        int x = scanner.nextInt();
        System.out.println("What is the height? ");
        int y = scanner.nextInt();

        System.out.println("Your rectangle has an area of: " + calculateArea(x, y));
        scanner.close();
    }
}
```

Next Task

Task ID: removeDeadCode_java

The function `calculateFinalScore` calculates a student's score in a course. The course is removing the bonus assignment, and the function needs to be updated. Remove all code that involves computing the bonus.

```
public class Main {
    static float calculateFinalScore(int homeworkGrade, int examGrade, boolean didBonus) {
        // Exam is worth 60%, homework is 40%
        float examScore = examGrade * 0.6f;
        float homeworkScore = homeworkGrade * 0.4f;

        // If the student did the bonus, boost their homework score by 10%
        if (didBonus) {
            homeworkScore = homeworkScore * 1.1f;
        }

        float classScore = examScore + homeworkScore;

        return classScore;
    }

    public static void main(String[] args) {
        System.out.println("Mary's final score is: " + calculateFinalScore(80, 70, false));
    }
}
```

Next Task

Task ID: addParameter_java

The `calculateDailyRevenue` function calculates the per-day revenue gained by a delivery company. The function is going to be expanded to allow the optional calculation of potential revenue if no orders were cancelled. Add a new boolean parameter to the function that allows for this upgrade to be made.

```
import java.util.ArrayList;
import java.util.List;

public class Main {
    static class Order {
        boolean cancelled;
        double price;
        int quantity;

        Order(boolean cancelled, double price, int quantity) {
            this.cancelled = cancelled;
            this.price = price;
            this.quantity = quantity;
        }
    }

    static double calculateDailyRevenue(List<Order> orders) {
        double total = 0.0;

        for (Order order : orders) {
            if (order.cancelled) {
                continue;
            }
            total += order.price * order.quantity;
        }

        return total;
    }

    public static void main(String[] args) {
        List<Order> sales = new ArrayList<>();

        Order staples = new Order(false, 5.65, 300);
        sales.add(staples);

        Order paperclips = new Order(true, 3.26, 150);
        sales.add(paperclips);

        System.out.println("Revenue: " + calculateDailyRevenue(sales));
    }
}
```

Next Task

Task ID: renameFunction_java

Rename the function `foo` to have a more appropriate name.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean foo(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (foo(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

Next Task

Task ID: commentCode_java

Write a comment above the `isStrongPassword` function that explains what it does along with the criteria for a strong password.

```
public static boolean isStrongPassword(String password) {
    if (password.length() < 8) {
        return false;
    }

    boolean hasUpper = false;
    boolean hasDigit = false;

    for (char ch : password.toCharArray()) {
        if (Character.isUpperCase(ch)) {
            hasUpper = true;
        }
        if (Character.isDigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

Next Task

Task ID: forToWhile_ListSearch_java

Convert the for loop in the following code to be a while loop.

```
public class ListTools {
    public static int searchList(int[] list, int key) {
        for (int i = 0; i < list.length; i++) {
            if (list[i] == key) {
                return i;
            }
        }
        return -1;
    }
}
```

Next Task

Task ID: functionsIntoClass_java

Create a new class called `Log` and encapsulate the following 3 functions into it as its methods. Make the `log` parameter a data member of the class and remove it from the parameter lists of the methods.

```
import java.io.PrintStream;

public class LogTools {

    static void writeLogEvent(PrintStream log, String msg) {
        log.println("[Info] " + msg);
    }

    static void writeLogError(PrintStream log, String msg) {
        log.println("[Error] " + msg);
    }

    static void closeLog(PrintStream log) {
        log.println("Log completed!");
    }
}
```

Next Task

Task ID: forToWhile_BubbleSort_java

Convert the for loops in the following code to be while loops.

```
public class ArrayTools {
    public static void bubbleSort(int[] list) {
        int n = list.length;
        boolean swapped;

        for (int i = 0; i < n - 1; ++i) {
            swapped = false;
            for (int j = 0; j < n - i - 1; ++j) {
                if (list[j] > list[j + 1]) {
                    int tmp = list[j];
                    list[j] = list[j + 1];
                    list[j + 1] = tmp;
                    swapped = true;
                }
            }

            // If no two elements were swapped, then break
            if (!swapped) {
                break;
            }
        }
    }
}
```

Next Task

Task ID: generalizeAvgFunctions_java

The following code contains two functions that average students' exam and homework scores. Replace the two specific functions with one general function that can work on both the exam and homework arrays. Move the printing to be in the main function.

```
import java.util.Arrays;
import java.util.List;

public class Main {
    // Compute and print the average of exam scores
    static void printExamAverage(List<Integer> scores) {
        int sum = 0;
        for (int s : scores) {
            sum += s;
        }
        double avg = (double) sum / scores.size();
        System.out.println("Exam average: " + avg);
    }

    // Compute and print the average of homework scores
    static void printHomeworkAverage(List<Integer> scores) {
        int sum = 0;
        for (int s : scores) {
            sum += s;
        }
        double avg = (double) sum / scores.size();
        System.out.println("Homework average: " + avg);
    }

    public static void main(String[] args) {
        List<Integer> exams = Arrays.asList(85, 90, 78);
        List<Integer> homework = Arrays.asList(100, 95, 88, 92);

        printExamAverage(exams);
        printHomeworkAverage(homework);
    }
}
```

Next Task

Task ID: implementFileSummation_java

Implement the `readAndSumNumbers` function. It should read two numbers from the provided file, print the sum of the two, then return the value.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Main {
    static int readAndSumNumbers(Scanner file) {

    }

    public static void main(String[] args) {
        try {
            Scanner numbersFile = new Scanner(new File("data.txt"));
            while (numbersFile.hasNextInt()) {
                readAndSumNumbers(numbersFile);
            }
            numbersFile.close();
        } catch (FileNotFoundException e) {
            System.exit(1);
        }
    }
}
```

Next Task

Python Tasks:

Task ID: nameChange_py

Change all the variables named `x` to be `width` and the variables named `y` to be `height`.

```
def calculateArea(x, y):
    return x * y

x = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(x,y))
```

Next Task

Task ID: removeDeadCode_py

The function `calculateFinalScore` calculates a student's score in a course. The course is removing the bonus assignment, and the function needs to be updated. Remove all code that involves computing the bonus.

```
def calculateFinalScore(homeworkGrade, examGrade, didBonus):
    # Exam is worth 60%, homework is 40%
    examScore = examGrade * 0.6
    homeworkScore = homeworkGrade * 0.4

    # If the student did the bonus, boost their homework score by 10%
    if didBonus:
        homeworkScore = homeworkScore * 1.1

    classScore = examScore + homeworkScore

    return classScore


print("Mary's final score is:", calculateFinalScore(80,70,False))
```

Next Task

Task ID: addParameter_py

The `calculateDailyRevenue` function calculates the per-day revenue gained by a delivery company. The function is going to be expanded to allow the optional calculation of potential revenue if no orders were cancelled. Add a new boolean parameter to the function that allows for this upgrade to be made.

```
class Order:
    def __init__(self, cancelled, price, quantity):
        self.cancelled = cancelled
        self.price = price
        self.quantity = quantity


def calculateDailyRevenue(orders):
    total = 0.0

    for order in orders:
        if order.cancelled:
            continue
        total += order.price * order.quantity

    return total


sales = []

staples = Order(False, 5.65, 300)
sales.append(staples)

paperclips = Order(True, 3.26, 150)
sales.append(paperclips)

print("Revenue:", calculateDailyRevenue(sales))
```

Next Task

Task ID: renameFunction_py

Rename the function `foo` to have a more appropriate name.

```
def foo(list, key):
    for i in range(len(list)):
        if list[i] == key:
            return True
    return False


data = [ 1, 4, 7, 11 ]

if foo(data, 6):
    print("6 is in the list.")
```

Next Task

Task ID: commentCode_py

Write a comment above the `isStrongPassword` function that explains what it does along with the criteria for a strong password.

```
def is_strong_password(password):
    if len(password) < 8:
        return False

    has_upper = False
    has_digit = False

    for char in password:
        if char.isupper():
            has_upper = True
        if char.isdigit():
            has_digit = True

    return has_upper and has_digit
```

Next Task

Task ID: forToWhile_ListSearch_py

Convert the for loop in the following code to be a while loop.

```
def searchList(list, key):
    for i in range(len(list)):
        if list[i] == key:
            return i
    return -1
```

Next Task

Task ID: functionsIntoClass_py

Create a new class called `Log` and encapsulate the following 3 functions into it as its methods. Make the `log` parameter a data member of the class and remove it from the parameter lists of the methods.

```
def writeLogEvent(log, msg):
    log.write("[Info] " + msg)

def writeLogError(log, msg):
    log.write("[Error] " + msg)

def closeLog(log):
    log.write("Log completed!")
```

Next Task

Task ID: forToWhile_BubbleSort_py

Convert the for loops in the following code to be while loops.

```
def bubbleSort(list):
    n = len(list)

    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

Next Task

Task ID: generalizeAvgFunctions_py

The following code contains two functions that average students' exam and homework scores. Replace the two specific functions with one general function that can work on both the exam and homework arrays. Move the printing to be in the main function.

```
# Compute and print the average of exam scores
def printExamAverage(scores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

# Compute and print the average of homework scores
def printHomeworkAverage(scores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

Next Task

Task ID: implementFileSummation_py

Implement the `readAndSumNumbers` function. It should read two numbers from the provided file, print the sum of the two, then return the value. There are only 2 numbers per line in the file, separated by a space.

```
def readAndSumNumbers(line):
    pass

numbers_file = open("data.txt",'r')

for line in numbers_file:
    readAndSumNumbers(line)

numbers_file.close()
```

Next Task

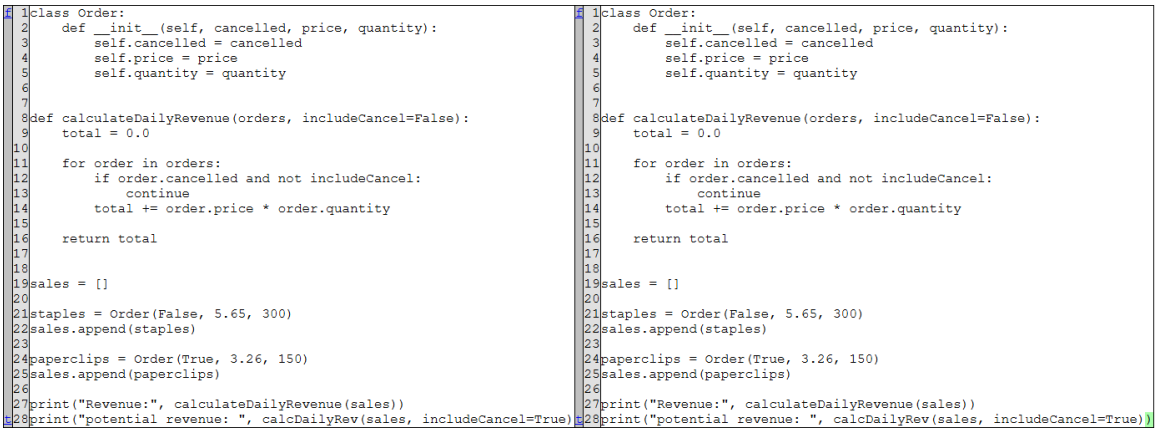
APPENDIX B

Source Code Edit Reference Set

The following is a complete list of all edits within the edit reference set. All edits are visualized using a line-based textual diffing tool.

Task: addParameter_1

Text events: 153-153

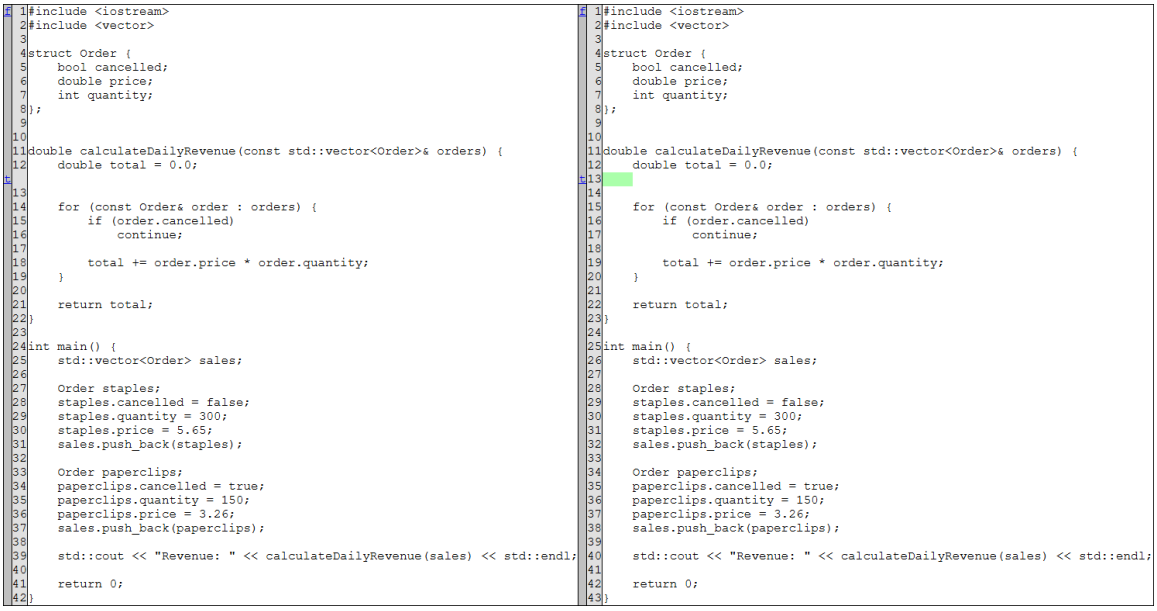


```
class Order:
    def __init__(self, cancelled, price, quantity):
        self.cancelled = cancelled
        self.price = price
        self.quantity = quantity


def calculateDailyRevenue(orders, includeCancel=False):
    total = 0.0

    for order in orders:
        if order.cancelled and not includeCancel:
            continue
        total += order.price * order.quantity

    return total


sales = []

staples = Order(False, 5.65, 300)
sales.append(staples)

paperclips = Order(True, 3.26, 150)
sales.append(paperclips)

print("Revenue:", calculateDailyRevenue(sales))
print("potential revenue: ", calcDailyRev(sales, includeCancel=True))
```

```
class Order:
    def __init__(self, cancelled, price, quantity):
        self.cancelled = cancelled
        self.price = price
        self.quantity = quantity


def calculateDailyRevenue(orders, includeCancel=False):
    total = 0.0

    for order in orders:
        if order.cancelled and not includeCancel:
            continue
        total += order.price * order.quantity

    return total


sales = []

staples = Order(False, 5.65, 300)
sales.append(staples)

paperclips = Order(True, 3.26, 150)
sales.append(paperclips)

print("Revenue:", calculateDailyRevenue(sales))
print("potential revenue: ", calcDailyRev(sales, includeCancel=True))
```

Task: addParameter_3

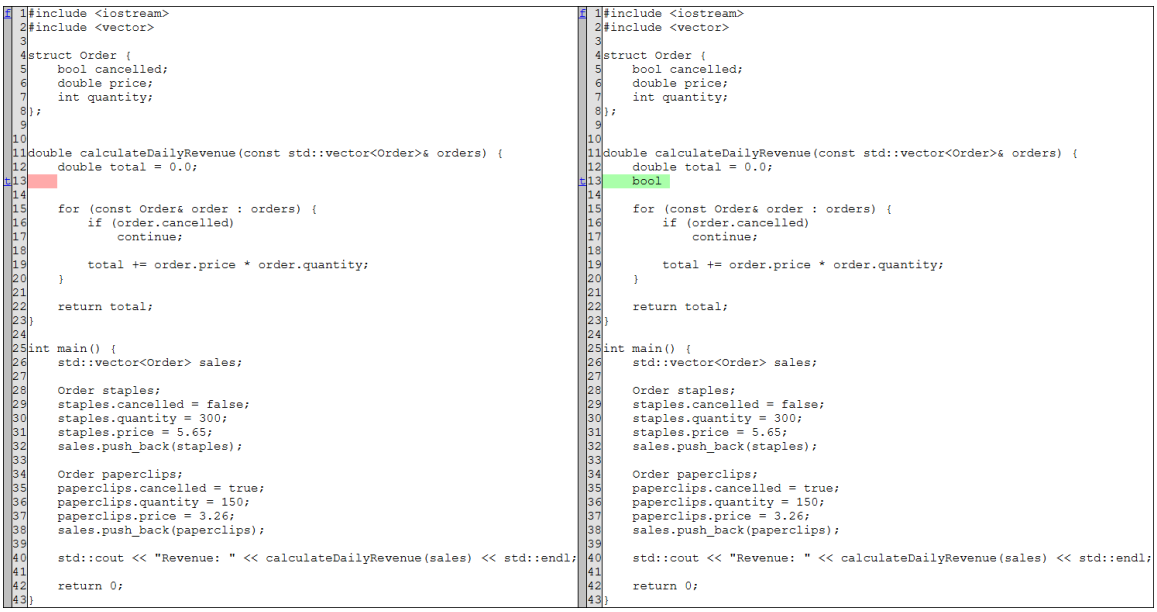
Text events: 1-5



```cpp
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders) {
    double total = 0.0;


    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

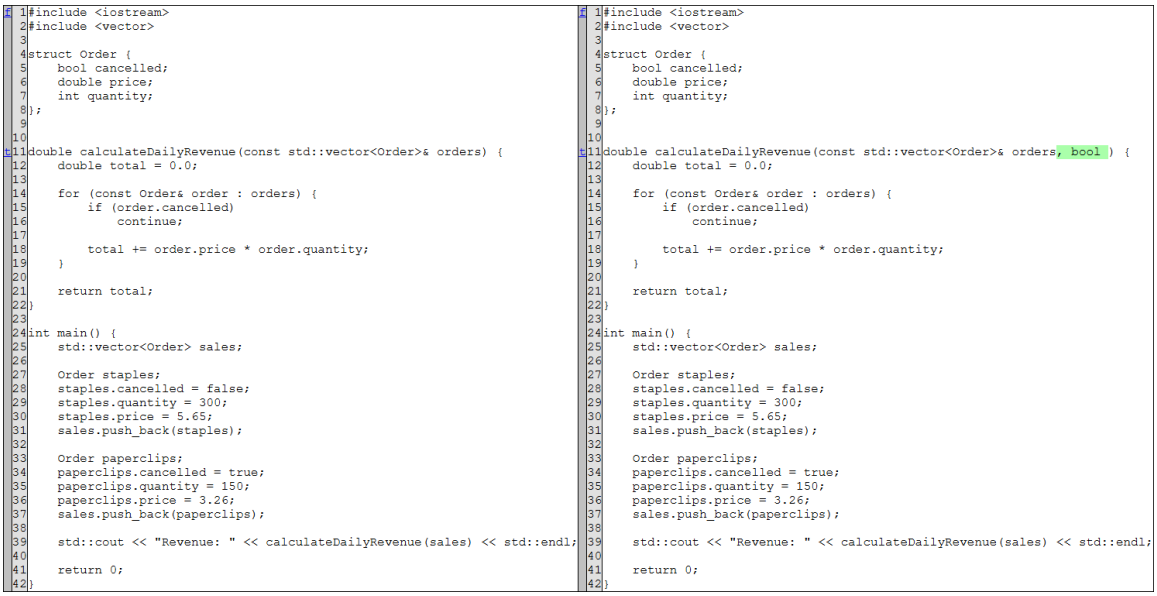
Text events: 6-10



```
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

```
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders) {
    double total = 0.0;
    bool

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

Text events: 21-27



```
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

```
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders, bool ) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

Text events: 28-45

```cpp
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders, bool ) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

struct Order {
    bool cancelled;
    double price;
    int quantity;
};


double calculateDailyRevenue(const std::vector<Order>& orders, bool noOrderRevenue) {
    double total = 0.0;

    for (const Order& order : orders) {
        if (order.cancelled)
            continue;

        total += order.price * order.quantity;
    }

    return total;
}

int main() {
    std::vector<Order> sales;

    Order staples;
    staples.cancelled = false;
    staples.quantity = 300;
    staples.price = 5.65;
    sales.push_back(staples);

    Order paperclips;
    paperclips.cancelled = true;
    paperclips.quantity = 150;
    paperclips.price = 3.26;
    sales.push_back(paperclips);

    std::cout << "Revenue: " << calculateDailyRevenue(sales) << std::endl;

    return 0;
}
```

Task: commentCode_1

Text events: 1-4

```cpp
#include <string>


bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

```cpp
#include <string>

//
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

Text events: 5-10

```
#include <string>

//
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

```
#include <string>

/* */
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

Text events: 80-81

```
#include <string>

/* Determins if a password is strong if the length is */
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

```
#include <string>

/* Determins if a password is strong if the length is */
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

Text events: 109-112

```
#include <string>

/* Determins if a password is strong if length > 8 */
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

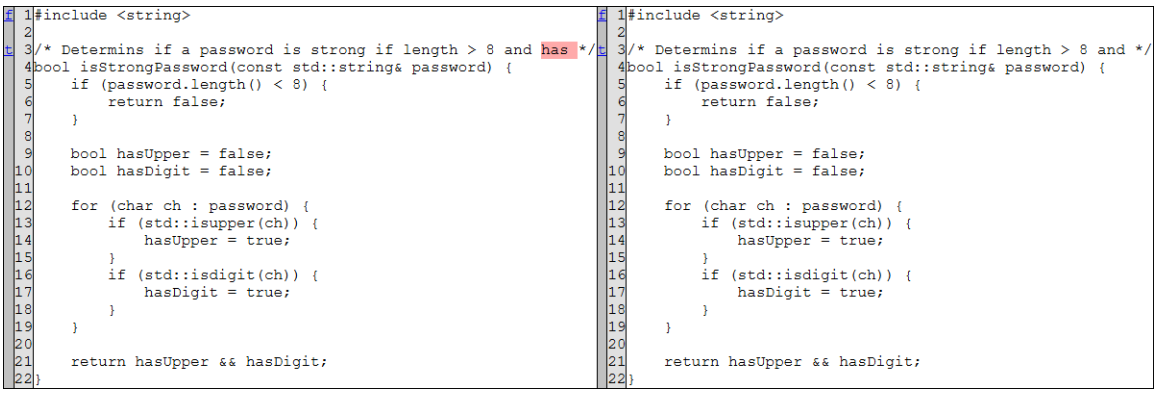
```
#include <string>

/* Determins if a password is strong if length > 8 and */
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

Text events: 112-116

```
#include <string>

/* Determins if a password is strong if length > 8 and*/
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

```
#include <string>

/* Determins if a password is strong if length > 8 and has */
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

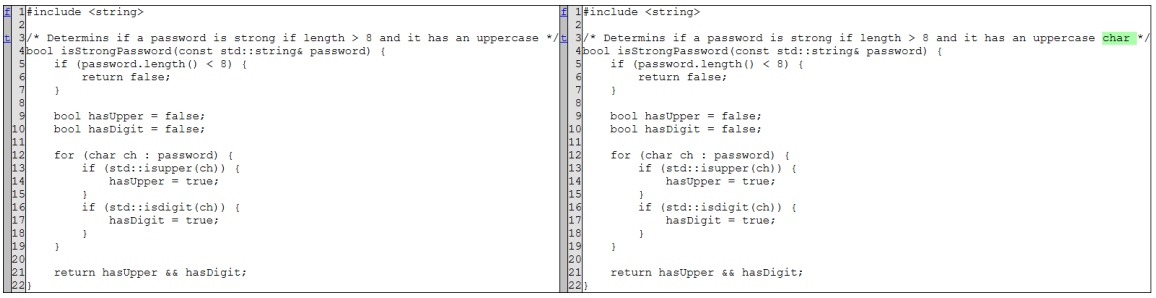
Text events: 117-120



```
#include <string>

/* Determins if a password is strong if length > 8 and has */
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

```
#include <string>

/* Determins if a password is strong if length > 8 and */
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

Text events: 141-145



```
#include <string>

/* Determins if a password is strong if length > 8 and it has an uppercase */
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

```
#include <string>

/* Determins if a password is strong if length > 8 and it has an uppercase char */
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

Task: commentCode_2

Text events: 7-19

```
#include <string>

//
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

```
#include <string>

// calculates
bool isStrongPassword(const std::string& password) {
    if (password.length() < 8) {
        return false;
    }

    bool hasUpper = false;
    bool hasDigit = false;

    for (char ch : password) {
        if (std::isupper(ch)) {
            hasUpper = true;
        }
        if (std::isdigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

Task: commentCode_3

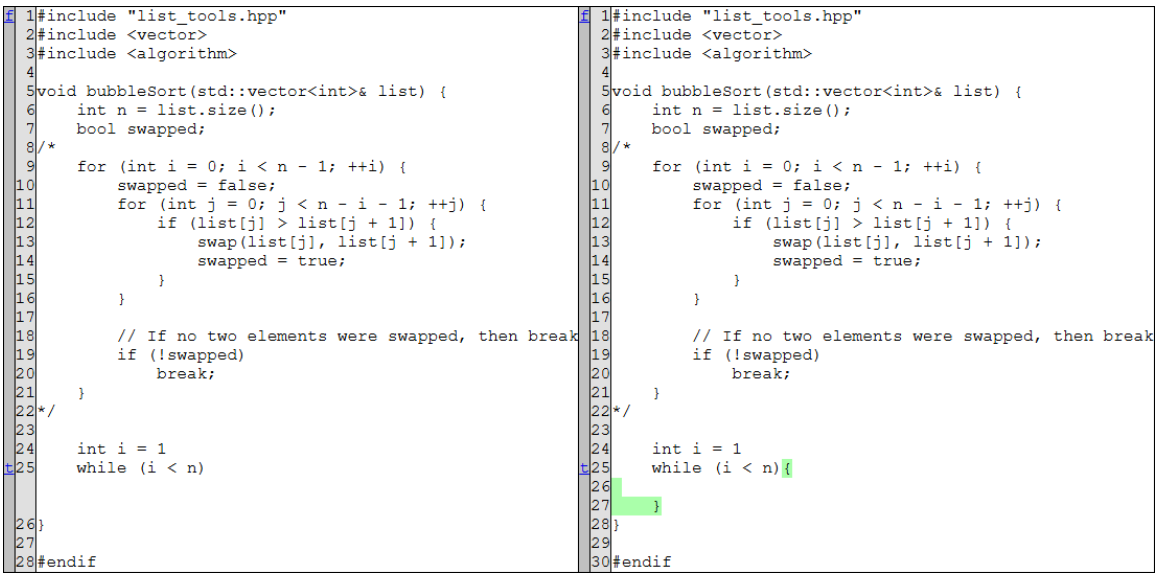
Text events: 1-18

```

public static boolean isStrongPassword(String password) {
    if (password.length() < 8) {
        return false;
    }

    boolean hasUpper = false;
    boolean hasDigit = false;

    for (char ch : password.toCharArray()) {
        if (Character.isUpperCase(ch)) {
            hasUpper = true;
        }
        if (Character.isDigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

```
// isStrongPass
public static boolean isStrongPassword(String password) {
    if (password.length() < 8) {
        return false;
    }

    boolean hasUpper = false;
    boolean hasDigit = false;

    for (char ch : password.toCharArray()) {
        if (Character.isUpperCase(ch)) {
            hasUpper = true;
        }
        if (Character.isDigit(ch)) {
            hasDigit = true;
        }
    }

    return hasUpper && hasDigit;
}
```

Task: forToWhile_BubbleSort_1

Text events: 31-34



```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/

    int i = 1
    while (i < n)



}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/

    int i = 1
    while (i < n){

    }
}

#endif
```

Text events: 91-92

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/

    int i = 1
    int j = 1
    while (i < n){
        swapped = false;
        while (j < n i - 1)
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/

    int i = 1;
    int j = 1;
    while (i < n){
        swapped = false;
        while (j < n i - 1)
    }
}

#endif
```

Text events: 92-93

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/

    int i = 1
    int j = 1;
    while (i < n){
        swapped = false;
        while (j < n i - 1)
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/
/

    int i = 1;
    int j = 1;
    while (i < n){
        swapped = false;
        while (j < n i - 1)
    }
}

#endif
```

Text events: 180-184

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/
// just remembered semicolons are a thing. Python has been too nice to me.

    int i = 1;
    int j = 1;
    while (i < n){
        swapped = false;
        while (j < n i - 1)



    }
}

#endif
```

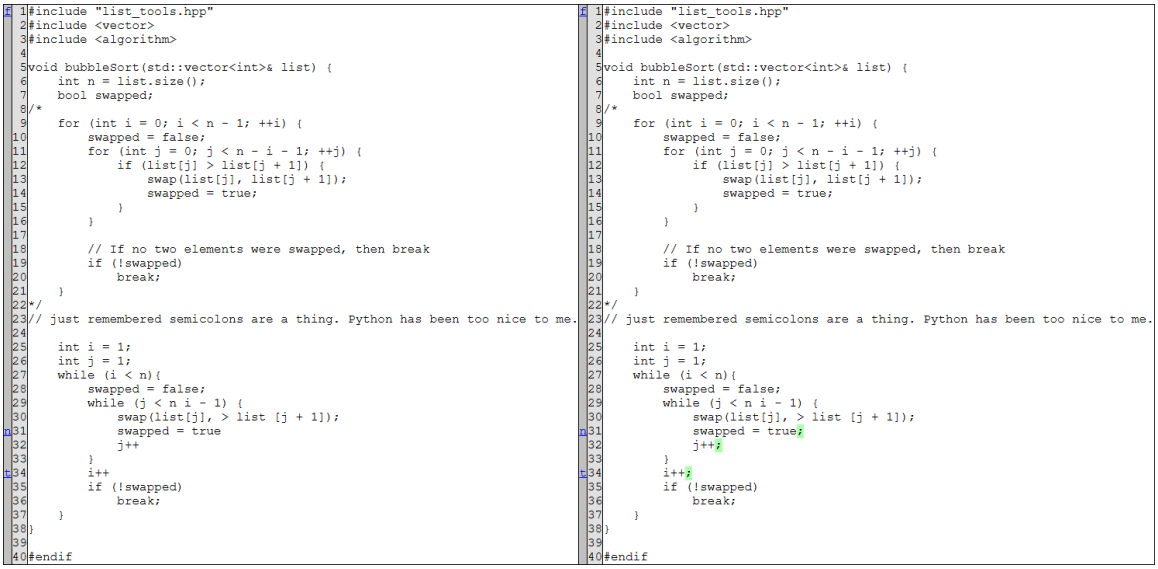
```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/
// just remembered semicolons are a thing. Python has been too nice to me.

    int i = 1;
    int j = 1;
    while (i < n){
        swapped = false;
        while (j < n i - 1) {


        }
    }
}

#endif
```

Text events: 244-247

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/
// just remembered semicolons are a thing. Python has been too nice to me.

    int i = 1;
    int j = 1;
    while (i < n){
        swapped = false;
        while (j < n i - 1) {
            swap(list[j], > list [j + 1]);
            swapped = true
            j++
        }


    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/
// just remembered semicolons are a thing. Python has been too nice to me.

    int i = 1;
    int j = 1;
    while (i < n){
        swapped = false;
        while (j < n i - 1) {
            swap(list[j], > list [j + 1]);
            swapped = true
            j++
        }
        i++
    }
}

#endif
```

Text events: 258-260

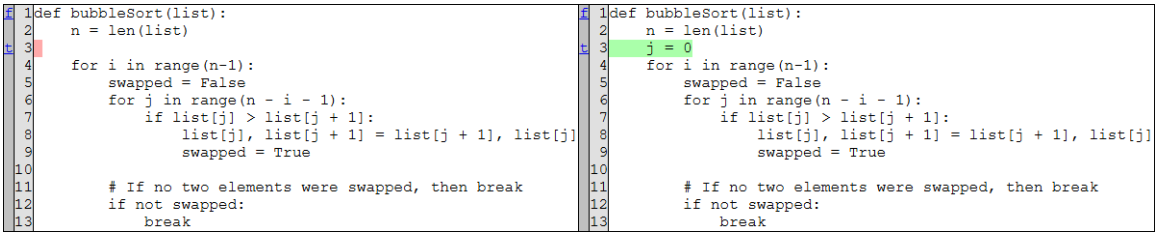


```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/
// just remembered semicolons are a thing. Python has been too nice to me.

    int i = 1;
    int j = 1;
    while (i < n){
        swapped = false;
        while (j < n i - 1) {
            swap(list[j], > list [j + 1]);
            swapped = true
            j++
        }
        i++
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;
/*
    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
*/
// just remembered semicolons are a thing. Python has been too nice to me.

    int i = 1;
    int j = 1;
    while (i < n){
        swapped = false;
        while (j < n i - 1) {
            swap(list[j], > list [j + 1]);
            swapped = true;
            j++;
        }
        i++;
        if (!swapped)
            break;
    }
}

#endif
```

Task: forToWhile_BubbleSort_2

Text events: 1-5



```
def bubbleSort(list):
    n = len(list)

    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

```
def bubbleSort(list):
    n = len(list)
    j = 0
    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

Text events: 2-37

```
def bubbleSort(list):
    n = len(list)
    j = 0
    while i < n-1:

    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

```
def bubbleSort(list):
    n = len(list)
    i = 0
    while i < n-1:

    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

Text events: 40-44

```
def bubbleSort(list):
    n = len(list)
    i = 0
    while i < n-1:

    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

```
def bubbleSort(list):
    n = len(list)
    i = 0
    while i < n-1:
        j = 0
    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

Text events: 98-143

```
def bubbleSort(list):
    n = len(list)
    i = 0
    while i < n-1:
        j = 0
        while j < n-i-1:
            if list[j] > list[j+1]:

    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

```
def bubbleSort(list):
    n = len(list)
    i = 0
    while i < n-1:
        j = 0
        while j < n-i-1:
            if list[j] > list[j+1]:
                list[j], list[j+1] = list[j+1], list[j]
    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

Text events: 159-170

```
def bubbleSort(list):
    n = len(list)
    i = 0
    while i < n-1:
        j = 0
        while j < n-i-1:
            if list[j] > list[j+1]:
                list[j], list[j+1] = list[j+1], list[j]
                swapped true
    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

```
def bubbleSort(list):
    n = len(list)
    i = 0
    while i < n-1:
        j = 0
        while j < n-i-1:
            if list[j] > list[j+1]:
                list[j], list[j+1] = list[j+1], list[j]
                swapped = True
    for i in range(n-1):
        swapped = False
        for j in range(n - i - 1):
            if list[j] > list[j + 1]:
                list[j], list[j + 1] = list[j + 1], list[j]
                swapped = True

        # If no two elements were swapped, then break
        if not swapped:
            break
```

Task: forToWhile_BubbleSort_3

Text events: 23-27

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    for (int i = 0; i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    for (i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 28-30

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    for (i < n - 1; ++i) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    for (i < n - 1) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 31-35

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    for (i < n - 1) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 36-49

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 50-50

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {

        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {

        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 51-54

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {

        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 55-59

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {

            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {

            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 60-63

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {

            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            ++j;
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 64-68

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        for (int j = 0; j < n - i - 1; ++j) {
            ++j;
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        while(int j = 0; j < n - i - 1; ++j) {
            ++j;
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 69-69

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        while(int j = 0; j < n - i - 1; ++j) {
            ++j;
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        while (int j = 0; j < n - i - 1; ++j) {
            ++j;
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 70-74

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        while (int j = 0; j < n - i - 1; ++j) {
            ++j;
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

```
#include "list_tools.hpp"
#include <vector>
#include <algorithm>

void bubbleSort(std::vector<int>& list) {
    int n = list.size();
    bool swapped;

    int i;
    int j;

    while(i < n - 1) {
        ++i;
        swapped = false;
        while (j < n - i - 1; ++j) {
            ++j;
            if (list[j] > list[j + 1]) {
                swap(list[j], list[j + 1]);
                swapped = true;
            }
        }

        // If no two elements were swapped, then break
        if (!swapped)
            break;
    }
}

#endif
```

Text events: 75-79

```
f  1#include "list_tools.hpp"
   2#include <vector>
   3#include <algorithm>
   4
   5void bubbleSort(std::vector<int>& list) {
   6    int n = list.size();
   7    bool swapped;
   8
   9    int i;
  10    int j;
  11
  12    while(i < n - 1) {
  13        ++i;
  14        swapped = false;
t 15        while (j < n - i - 1; ++j) {
  16            ++j;
  17            if (list[j] > list[j + 1]) {
  18                swap(list[j], list[j + 1]);
  19                swapped = true;
  20            }
  21        }
  22
  23        // If no two elements were swapped, then break
  24        if (!swapped)
  25            break;
  26    }
  27}
  28
  29#endif
```

```
f  1#include "list_tools.hpp"
   2#include <vector>
   3#include <algorithm>
   4
   5void bubbleSort(std::vector<int>& list) {
   6    int n = list.size();
   7    bool swapped;
   8
   9    int i;
  10    int j;
  11
  12    while(i < n - 1) {
  13        ++i;
  14        swapped = false;
t 15        while (j < n - i - 1) {
  16            ++j;
  17            if (list[j] > list[j + 1]) {
  18                swap(list[j], list[j + 1]);
  19                swapped = true;
  20            }
  21        }
  22
  23        // If no two elements were swapped, then break
  24        if (!swapped)
  25            break;
  26    }
  27}
  28
  29#endif
```

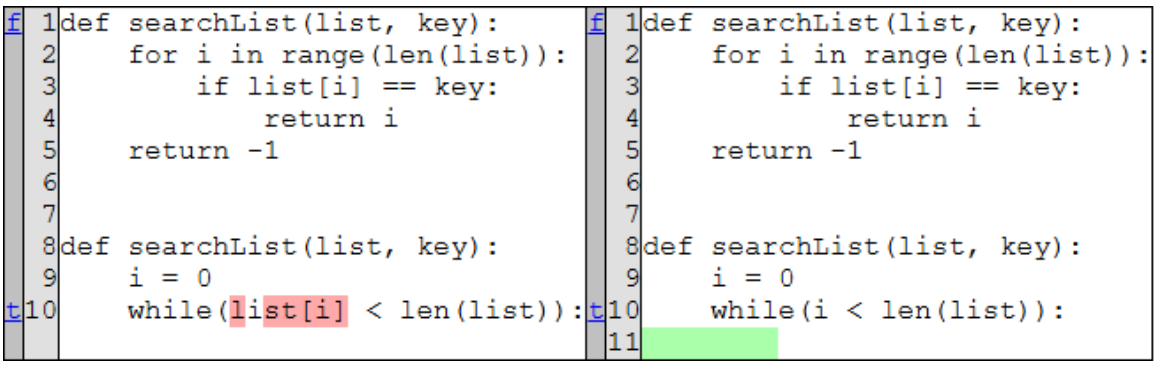
Task: forToWhile_ListSearch_1

Text events: 9-15

```
f 1def searchList(list, key):
  2    for i in range(len(list)):
  3        if list[i] == key:
  4            return i
  5    return -1
  6
  7
  8def searchList(list, key):
t
```

```
f 1def searchList(list, key):
  2    for i in range(len(list)):
  3        if list[i] == key:
  4            return i
  5    return -1
  6
  7
  8def searchList(list, key):
t 9    while()
```

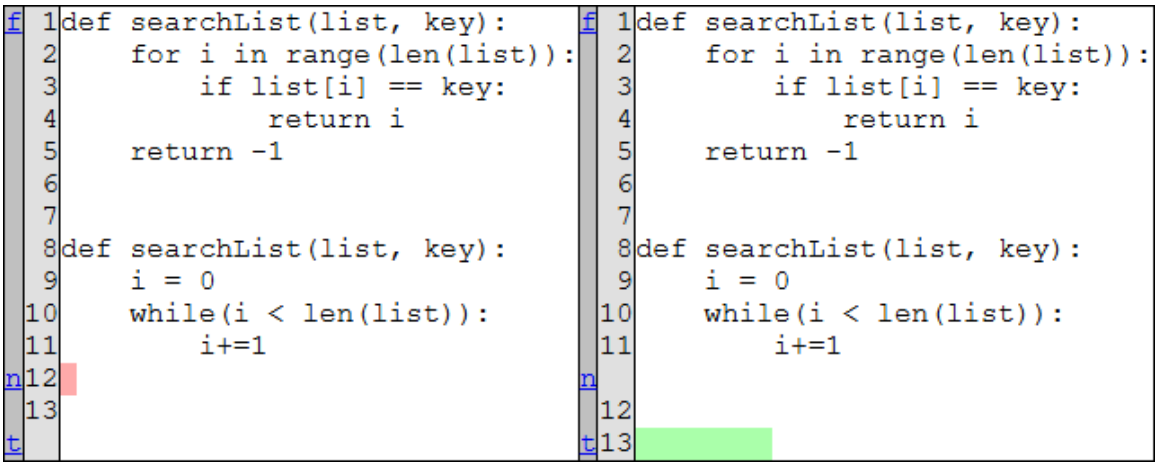
Text events: 59-66



```
def searchList(list, key):
    for i in range(len(list)):
        if list[i] == key:
            return i
    return -1


def searchList(list, key):
    i = 0
    while(list[i] < len(list)):
```

```
def searchList(list, key):
    for i in range(len(list)):
        if list[i] == key:
            return i
    return -1


def searchList(list, key):
    i = 0
    while(i < len(list)):
```

Text events: 75-78



```
def searchList(list, key):
    for i in range(len(list)):
        if list[i] == key:
            return i
    return -1


def searchList(list, key):
    i = 0
    while(i < len(list)):
        i+=1

```

```
def searchList(list, key):
    for i in range(len(list)):
        if list[i] == key:
            return i
    return -1


def searchList(list, key):
    i = 0
    while(i < len(list)):
        i+=1
```

Text events: 123-124

```
def searchList(list, key):
    for i in range(len(list)):
        if list[i] == key:
            return i
    return -1


def searchList(list, key):
    i = 0
    while(i < len(list)):
        if list[i] == key:
            return i
        i+=1
    return -1
```

```
def searchList(list, key):
    for i in range(len(list)):
        if list[i] == key:
            return i
    return -1


def searchList(list, key):
    i = 0
    while(i < len(list)-1):
        if list[i] == key:
            return i
        i+=1
    return -1
```

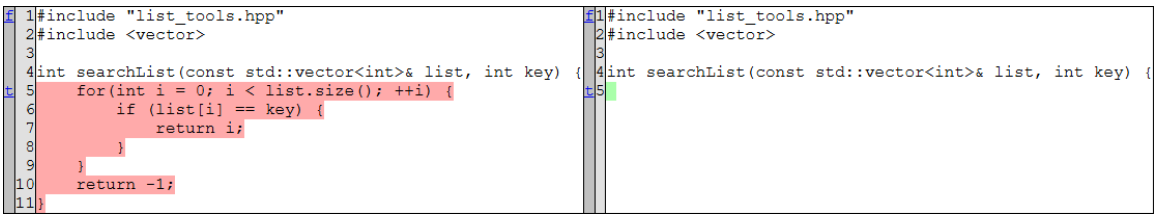
Text events: 125-125

```
def searchList(list, key):
    for i in range(len(list)):
        if list[i] == key:
            return i
    return -1


def searchList(list, key):
    i = 0
    while(i < len(list)-1):
        if list[i] == key:
            return i
        i+=1
    return -1
```

```
def searchList(list, key):
    i = 0
    while(i < len(list)-1):
        if list[i] == key:
            return i
        i+=1
    return -1
```

Task: forToWhile_ListSearch_2

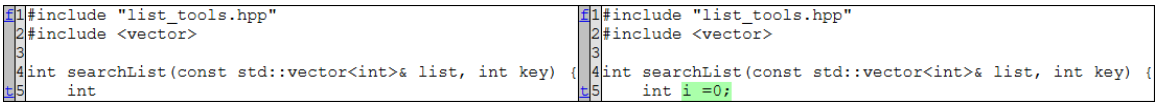
Text events: 1-2



```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {

```

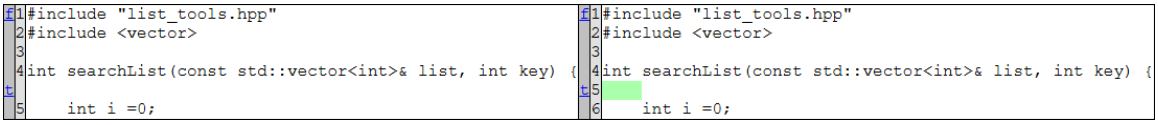
Text events: 16-22



```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int i =0;
```

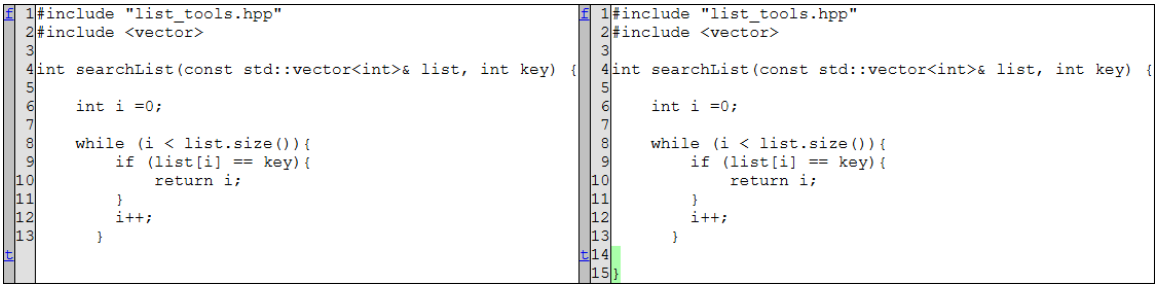
Text events: 23-28



```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {

    int i =0;
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {

    int i =0;
```

Text events: 162-164

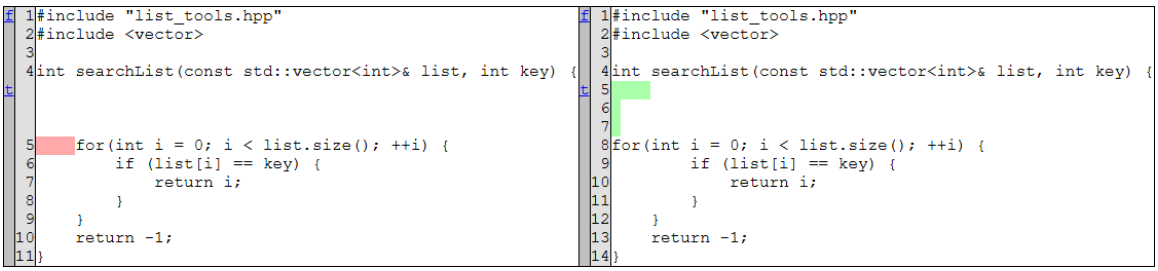


```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {

    int i =0;

    while (i < list.size()){
        if (list[i] == key){
            return i;
        }
        i++;
      }

```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {

    int i =0;

    while (i < list.size()){
        if (list[i] == key){
            return i;
        }
        i++;
      }

}
```

Task: forToWhile_ListSearch_3

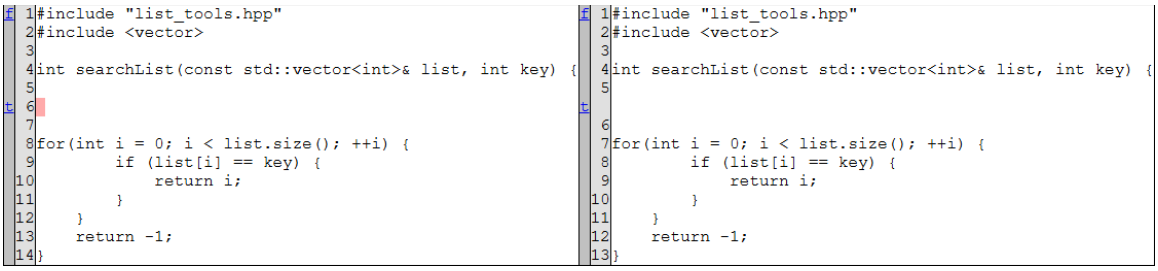
Text events: 1-3



```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

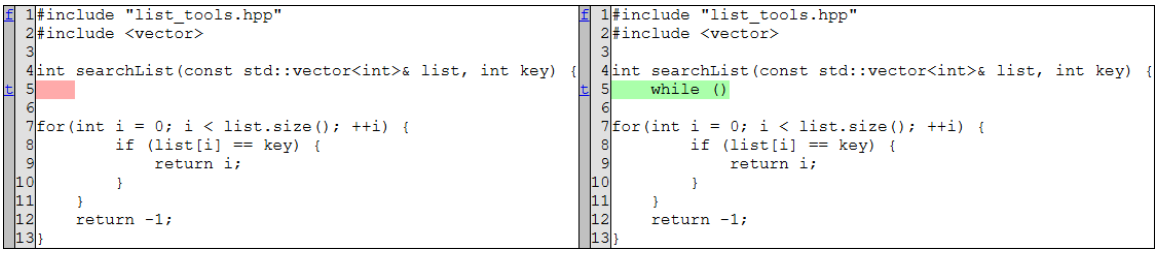
Text events: 4-12



```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {


for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {


for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 13-20



```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    while ()
for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 21-21

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {

    while ()

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {

while ()

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 22-34

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {

while ()

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
while ()

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 58-62

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size())




for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {

}

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 63-67

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {

}


for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if ()
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 78-82

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[])
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index])
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 83-89

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index])
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index] == key)
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 90-94

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index] == key)



    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index] == key) {

}
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 95-106

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index] == key) {

}
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index] == key) {
            return index
        }
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 107-107

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index] == key) {
            return index
        }
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index] == key) {
            return index;
        }
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 108-111

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index] == key) {
            return index;
        }
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index] == key) {
            return index;
        }
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 112-112

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0
    while (index < list.size()) {
        if (list[index] == key) {
            return index;
        }
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0;
    while (index < list.size()) {
        if (list[index] == key) {
            return index;
        }
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Text events: 113-114

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0;
    while (index < list.size()) {
        if (list[index] == key) {
            return index;
        }
    }

for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return i;
        }
    }
    return -1;
}
```

```
#include "list_tools.hpp"
#include <vector>

int searchList(const std::vector<int>& list, int key) {
    int index = 0;
    while (index < list.size()) {
        if (list[index] == key) {
            return index;
        }
    }

    return -1;
}
```

Task: functionsIntoClass_1

Text events: 1-2

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}



#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}


#endif
```

Text events: 36-40

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:


private:

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:


private:

};

#endif
```

Text events: 59-60

```
 1#include <iostream>
 2#include <string>
 3
 4#ifndef LOG_HPP
 5#define LOG_HPP
 6
 7void writeLogEvent(std::ostream& log, const std::string& msg) {
 8    log << "[Info] " << msg << std::endl;
 9}
10
11void writeLogError(std::ostream& log, const std::string& msg) {
12    log << "[Error] " << msg << std::endl;
13}
14
15void closeLog(std::ostream& log) {
16    log << "Log completed!" << std::endl;
17}
18
19class Log{
20
21public:
22    void writeLogEvent
23
24private:
25
26};
27
28#endif
```

```
 1#include <iostream>
 2#include <string>
 3
 4#ifndef LOG_HPP
 5#define LOG_HPP
 6
 7void writeLogEvent(std::ostream& log, const std::string& msg) {
 8    log << "[Info] " << msg << std::endl;
 9}
10
11void writeLogError(std::ostream& log, const std::string& msg) {
12    log << "[Error] " << msg << std::endl;
13}
14
15void closeLog(std::ostream& log) {
16    log << "Log completed!" << std::endl;
17}
18
19class Log{
20
21public:
22    void writeLogEvent()
23
24private:
25
26};
27
28#endif
```

Text events: 83-84

```
 1#include <iostream>
 2#include <string>
 3
 4#ifndef LOG_HPP
 5#define LOG_HPP
 6
 7void writeLogEvent(std::ostream& log, const std::string& msg) {
 8    log << "[Info] " << msg << std::endl;
 9}
10
11void writeLogError(std::ostream& log, const std::string& msg) {
12    log << "[Error] " << msg << std::endl;
13}
14
15void closeLog(std::ostream& log) {
16    log << "Log completed!" << std::endl;
17}
18
19class Log{
20
21public:
22    void writeLogEvent(const std::string& msg)
23
24private:
25
26};
27
28#endif
```

```
 1#include <iostream>
 2#include <string>
 3
 4#ifndef LOG_HPP
 5#define LOG_HPP
 6
 7void writeLogEvent(std::ostream& log, const std::string& msg) {
 8    log << "[Info] " << msg << std::endl;
 9}
10
11void writeLogError(std::ostream& log, const std::string& msg) {
12    log << "[Error] " << msg << std::endl;
13}
14
15void closeLog(std::ostream& log) {
16    log << "Log completed!" << std::endl;
17}
18
19class Log{
20
21public:
22    void writeLogEvent(const std::string& msg){}
23
24private:
25
26};
27
28#endif
```

Text events: 85-89

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg){}

private:


};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg){}

private:


};

#endif
```

Text events: 114-128

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg){}



private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {


    }

private:
    const std::string msg;

};

#endif
```

Text events: 129-135

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {
    }

private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {
        log <<
    }

private:
    const std::string msg;

};

#endif
```

Text events: 135-142

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {
        log <<
    }

private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {
    }

private:
    const std::string msg;

};

#endif
```

Text events: 143-144

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {

    }


private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

private:
    const std::string msg;

};

#endif
```

Text events: 144-149

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }


private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }


private:
    const std::string msg;

};

#endif
```

Text events: 166-166

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {



private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}



private:
    const std::string msg;

};

#endif
```

Text events: 167-170

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}


private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}


private:
    const std::string msg;

};

#endif
```

Text events: 170-171

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}

private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(std::ostream& log, const std::string& msg)

private:
    const std::string msg;

};

#endif
```

Text events: 172-173

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(std::ostream& log, const std::string& msg)

private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg)

private:
    const std::string msg;

};

#endif
```

Text events: 174-176

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg)



private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg) {}


private:
    const std::string msg;

};

#endif
```

Text events: 176-180

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg) {}


private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg) {}


private:
    const std::string msg;

};

#endif
```

Text events: 181-185

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg) {}




private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg) {}


private:
    const std::string msg;

};

#endif
```

Text events: 232-239

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg) {}
    void closeLog(std::ostream& log) {}



private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg) {}
    void closeLog     (std::ostream& log) {}


private:
    const std::string msg;

};

#endif
```

Text events: 239-243

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg) {}
    void closeLog     (std::ostream& log) {}



private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg) {}
    void closeLog     (std::ostream& log)      {}



private:
    const std::string msg;

};

#endif
```

Text events: 143-244

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

public:
    void writeLogEvent(const std::string& msg) {}
    void writeLogError(const std::string& msg) {}
    void closeLog     (std::ostream& log)      {}



private:
    const std::string msg;

};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}
void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{

};

#endif
```

Text events: 267-272

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:



private:
};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:


private:
};

#endif
```

Text events: 382-388

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:
    Log(std::ostream& out) : log(out) {}

    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }



private:
};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:
    Log(std::ostream& out) : log(out) {}

    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }


private:
};

#endif
```

Text events: 388-389

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:
    Log(std::ostream& out) : log(out) {}

    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }


private:
};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:
    Log(std::ostream& out) : log(out) {}

    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}


private:
};

#endif
```

Text events: 405-406

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:
    Log(std::ostream& out) : log(out) {}

    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }


private:
};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:
    Log(std::ostream& out) : log(out) {}

    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}


private:
};

#endif
```

Text events: 438-439

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:
    Log(std::ostream& out) : log(out) {}

    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog(std::ostream& log) {
        log << "Log completed!" << std::endl;
    }


private:
    std::ostream& log;
};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:
    Log(std::ostream& out) : log(out) {}

    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog(std::ostream& log) {
        log << "Log completed!" << std::endl;
    }


private:
    std::ostream& log;
};

#endif
```

Text events: 440-442

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

void writeLogEvent(std::ostream& log, const std::string& msg) {
    log << "[Info] " << msg << std::endl;
}

void writeLogError(std::ostream& log, const std::string& msg) {
    log << "[Error] " << msg << std::endl;
}

void closeLog(std::ostream& log) {
    log << "Log completed!" << std::endl;
}

class Log{
public:
    Log(std::ostream& out) : log(out) {}

    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog(std::ostream& log) {
        log << "Log completed!" << std::endl;
    }


private:
    std::ostream& log;
};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP



class Log{
public:
    Log(std::ostream& out) : log(out) {}

    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog(std::ostream& log) {
        log << "Log completed!" << std::endl;
    }


private:
    std::ostream& log;
};

#endif
```

Task: functionsIntoClass_2

Text events: 28-36

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

class log {
public:
    void writeLogEvent(std::ostream& log, const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog(std::ostream& log) {
        log << "Log completed!" << std::endl;
    }



};

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

class log {
public:
    void writeLogEvent(std::ostream& log, const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog(std::ostream& log) {
        log << "Log completed!" << std::endl;
    }
private:


};

#endif
```

Text events: 58-58

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

class log {
public:
    void writeLogEvent(std::ostream& log, const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog(std::ostream& log) {
        log << "Log completed!" << std::endl;
    }
private:
    std::ostream log;
}

#endif
```

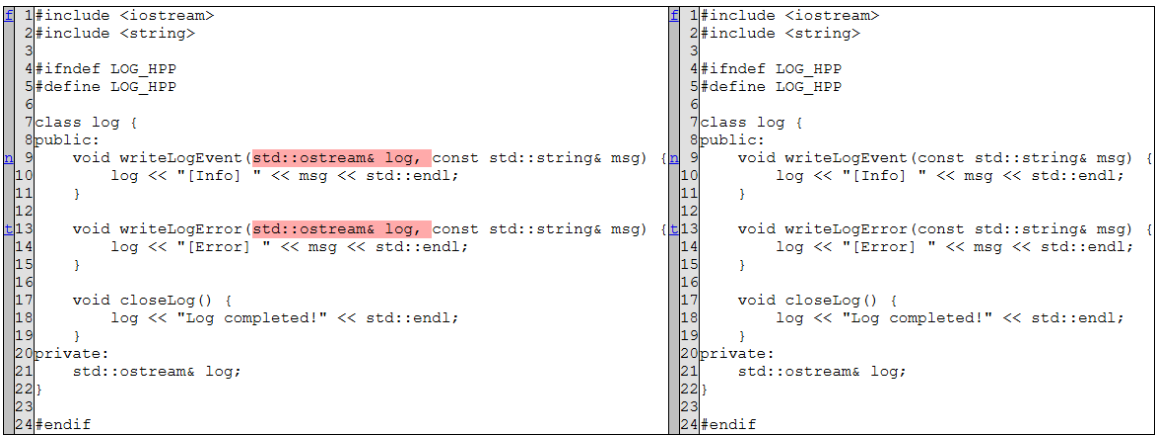
```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

class log {
public:
    void writeLogEvent(std::ostream& log, const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog() {
        log << "Log completed!" << std::endl;
    }
private:
    std::ostream log;
}

#endif
```

Text events: 59-59

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

class log {
public:
    void writeLogEvent(std::ostream& log, const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog() {
        log << "Log completed!" << std::endl;
    }
private:
    std::ostream log;
}

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

class log {
public:
    void writeLogEvent(std::ostream& log, const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog() {
        log << "Log completed!" << std::endl;
    }
private:
    std::ostream& log;
}

#endif
```

Text events: 60-61

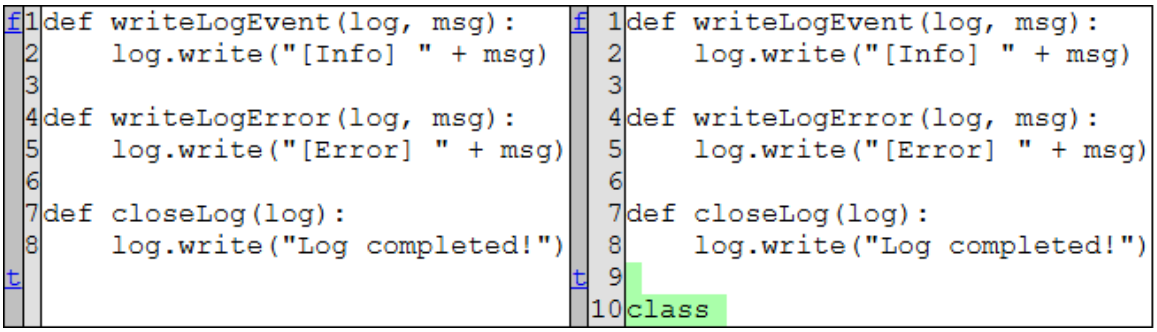


```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

class log {
public:
    void writeLogEvent(std::ostream& log, const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(std::ostream& log, const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog() {
        log << "Log completed!" << std::endl;
    }
private:
    std::ostream& log;
}

#endif
```

```
#include <iostream>
#include <string>

#ifndef LOG_HPP
#define LOG_HPP

class log {
public:
    void writeLogEvent(const std::string& msg) {
        log << "[Info] " << msg << std::endl;
    }

    void writeLogError(const std::string& msg) {
        log << "[Error] " << msg << std::endl;
    }

    void closeLog() {
        log << "Log completed!" << std::endl;
    }
private:
    std::ostream& log;
}

#endif
```

Task: functionsIntoClass_3

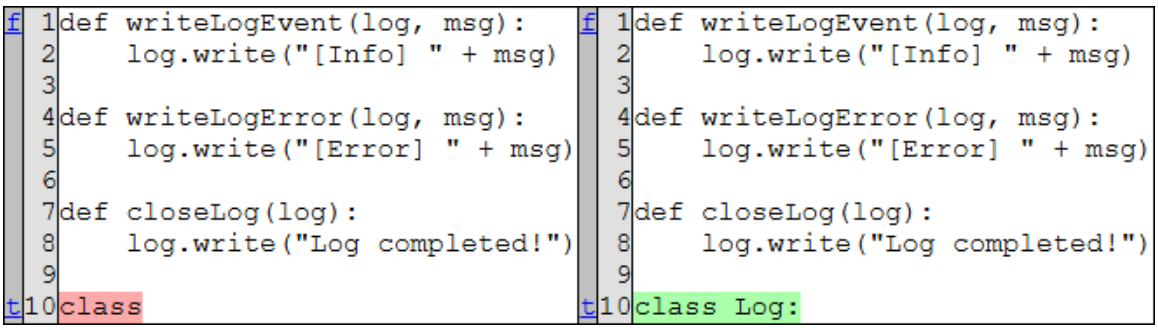
Text events: 1-7



```
def writeLogEvent(log, msg):
    log.write("[Info] " + msg)

def writeLogError(log, msg):
    log.write("[Error] " + msg)

def closeLog(log):
    log.write("Log completed!")
```

```
def writeLogEvent(log, msg):
    log.write("[Info] " + msg)

def writeLogError(log, msg):
    log.write("[Error] " + msg)

def closeLog(log):
    log.write("Log completed!")

class
```

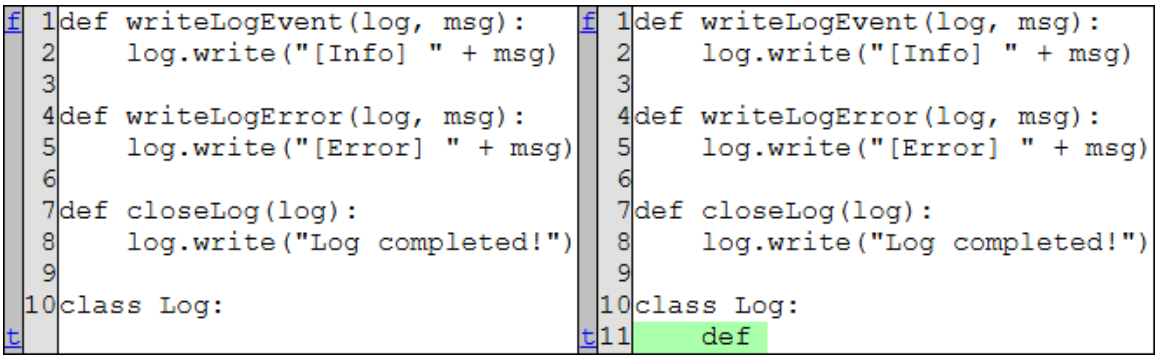
Text events: 7-12



```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
t 10 class
```

```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
t 10 class Log:
```

Text events: 13-16



```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
  10 class Log:
t
```

```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
  10 class Log:
t 11     def
```

Text events: 34-35

```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
  10 class Log:
t 11     def __init__(self):
```

```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
  10 class Log:
t 11     def __init__(self,):
  12
```

Text events: 36-39

```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
  10 class Log:
t 11     def __init__(self,):
  12
```

```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
  10 class Log:
t 11     def __init__(self, log):
  12
```

Text events: 40-48

```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
  10 class Log:
  11     def __init__(self, log):
t 12
```

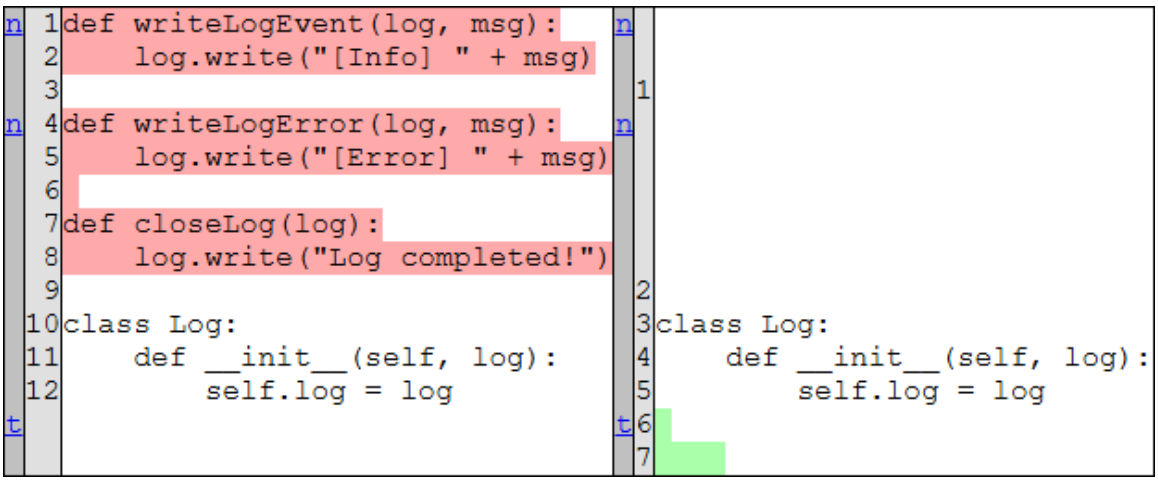
```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
  10 class Log:
  11     def __init__(self, log):
t 12         self.log
```

Text events: 49-55

```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
  10 class Log:
  11     def __init__(self, log):
t 12         self.log
```

```
f  1 def writeLogEvent(log, msg):
   2     log.write("[Info] " + msg)
   3
   4 def writeLogError(log, msg):
   5     log.write("[Error] " + msg)
   6
   7 def closeLog(log):
   8     log.write("Log completed!")
   9
  10 class Log:
  11     def __init__(self, log):
t 12         self.log = log
  13
```

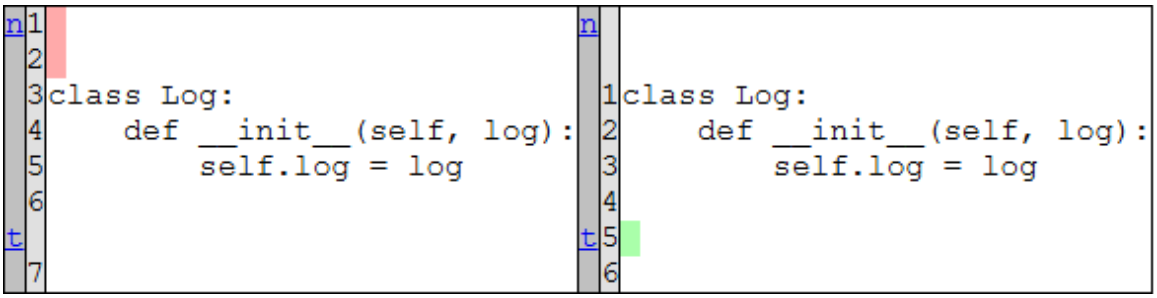
Text events: 55-56



```
def writeLogEvent(log, msg):
    log.write("[Info] " + msg)

def writeLogError(log, msg):
    log.write("[Error] " + msg)

def closeLog(log):
    log.write("Log completed!")

class Log:
    def __init__(self, log):
        self.log = log
```

```

class Log:
    def __init__(self, log):
        self.log = log
```

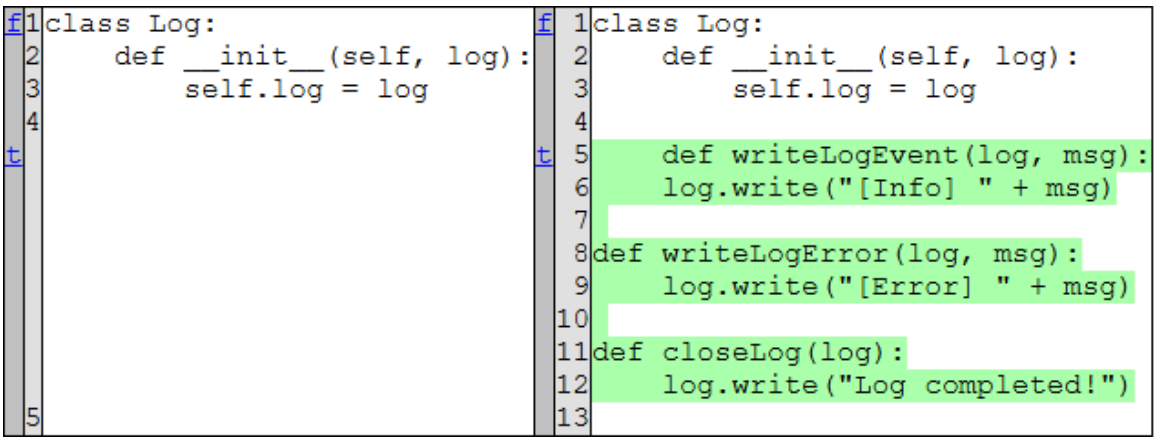
Text events: 57-59



```

class Log:
    def __init__(self, log):
        self.log = log
```

```
class Log:
    def __init__(self, log):
        self.log = log
```

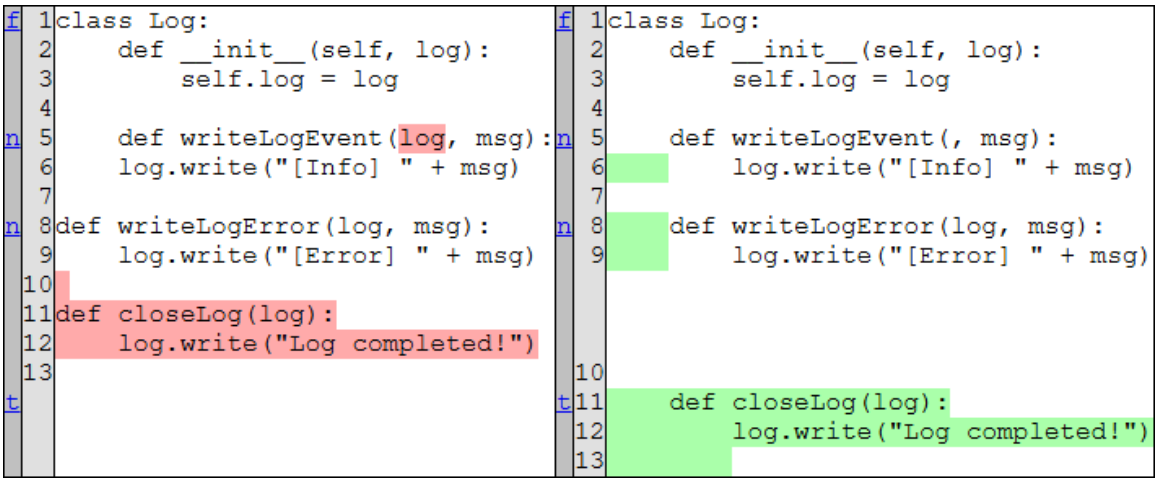
Text events: 59-60



```
class Log:
    def __init__(self, log):
        self.log = log
```

```
class Log:
    def __init__(self, log):
        self.log = log

    def writeLogEvent(log, msg):
    log.write("[Info] " + msg)

def writeLogError(log, msg):
    log.write("[Error] " + msg)

def closeLog(log):
    log.write("Log completed!")
```

Text events: 61-61



```
class Log:
    def __init__(self, log):
        self.log = log

    def writeLogEvent(log, msg):
    log.write("[Info] " + msg)

def writeLogError(log, msg):
    log.write("[Error] " + msg)

def closeLog(log):
    log.write("Log completed!")
```

```
class Log:
    def __init__(self, log):
        self.log = log

    def writeLogEvent(, msg):
        log.write("[Info] " + msg)

    def writeLogError(log, msg):
        log.write("[Error] " + msg)

    def closeLog(log):
        log.write("Log completed!")
```

Text events: 62-63

```
f  1 class Log:
   2     def __init__(self, log):
   3         self.log = log
   4
t  5     def writeLogEvent(, msg):
   6         log.write("[Info] " + msg)
   7
   8     def writeLogError(log, msg):
   9         log.write("[Error] " + msg)
  10
  11     def closeLog(log):
  12         log.write("Log completed!")
  13
```

```
f  1 class Log:
   2     def __init__(self, log):
   3         self.log = log
   4
t  5     def writeLogEvent(msg):
   6         log.write("[Info] " + msg)
   7
   8     def writeLogError(log, msg):
   9         log.write("[Error] " + msg)
  10
  11     def closeLog(log):
  12         log.write("Log completed!")
  13
```

Text events: 75-77

```
f  1 class Log:
   2     def __init__(self, log):
   3         self.log = log
   4
   5     def writeLogEvent(msg):
   6         self.log.write("[Info] " + msg)
   7
t  8     def writeLogError(log, msg):
   9         log.write("[Error] " + msg)
  10
  11     def closeLog(log):
  12         log.write("Log completed!")
  13
```

```
f  1 class Log:
   2     def __init__(self, log):
   3         self.log = log
   4
   5     def writeLogEvent(msg):
   6         self.log.write("[Info] " + msg)
   7
t  8     def writeLogError(msg):
   9         log.write("[Error] " + msg)
  10
  11     def closeLog(log):
  12         log.write("Log completed!")
  13
```

Text events: 78-82

```
f  1 class Log:
   2     def __init__(self, log):
   3         self.log = log
   4
   5     def writeLogEvent(msg):
   6         self.log.write("[Info] " + msg)
   7
   8     def writeLogError(msg):
t  9         log.write("[Error] " + msg)
  10
  11     def closeLog(log):
  12         log.write("Log completed!")
  13
```

```
f  1 class Log:
   2     def __init__(self, log):
   3         self.log = log
   4
   5     def writeLogEvent(msg):
   6         self.log.write("[Info] " + msg)
   7
   8     def writeLogError(msg):
t  9         self.log.write("[Error] " + msg)
  10
  11     def closeLog(log):
  12         log.write("Log completed!")
  13
```

Text events: 83-83

```
f  1 class Log:
   2     def __init__(self, log):
   3         self.log = log
   4
   5     def writeLogEvent(msg):
   6         self.log.write("[Info] " + msg)
   7
   8     def writeLogError(msg):
   9         self.log.write("[Error] " + msg)
  10
t 11     def closeLog(log):
  12         log.write("Log completed!")
  13
```

```
f  1 class Log:
   2     def __init__(self, log):
   3         self.log = log
   4
   5     def writeLogEvent(msg):
   6         self.log.write("[Info] " + msg)
   7
   8     def writeLogError(msg):
   9         self.log.write("[Error] " + msg)
  10
t 11     def closeLog():
  12         log.write("Log completed!")
  13
```

Text events: 84-88

```
f  1 class Log:
   2     def __init__(self, log):
   3         self.log = log
   4
   5     def writeLogEvent(msg):
   6         self.log.write("[Info] " + msg)
   7
   8     def writeLogError(msg):
   9         self.log.write("[Error] " + msg)
  10
  11     def closeLog():
t 12         log.write("Log completed!")
  13
```

```
f  1 class Log:
   2     def __init__(self, log):
   3         self.log = log
   4
   5     def writeLogEvent(msg):
   6         self.log.write("[Info] " + msg)
   7
   8     def writeLogError(msg):
   9         self.log.write("[Error] " + msg)
  10
  11     def closeLog():
t 12         self.log.write("Log completed!")
  13
```

Task:generalizeAvgFunctions_1

Text events: 1-6

```
# Compute and print the average of exam scores
def printExamAverage(scores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

# Compute and print the average of homework scores
def printHomeworkAverage(scores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

```
# Compute and print the average of exam scores
def printExamAverage(examScores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

# Compute and print the average of homework scores
def printHomeworkAverage(scores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

Text events: 7-11

```
# Compute and print the average of exam scores
def printExamAverage(examScores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

# Compute and print the average of homework scores
def printHomeworkAverage(scores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

```
# Compute and print the average of exam scores
def printExamAverage(examScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

# Compute and print the average of homework scores
def printHomeworkAverage(scores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

Text events: 41-42

```
f  1# Compute and print the average of exam and homework scores
   2def printExamAverage(examScores):
   3    sum = 0
   4    for s in examScores:
   5        sum += s
   6    avg = sum / len(scores)
   7    print("Exam average:", avg)
   8
t  9# Compute and print the average of homework scores
  10def printHomeworkAverage(scores):
  11    sum = 0
  12    for s in scores:
  13        sum += s
  14    avg = sum / len(scores)
  15    print("Homework average:", avg)
  16
  17exams = [ 85, 90, 78 ]
  18homework = [ 100, 95, 88, 92 ]
  19
  20printExamAverage(exams)
  21printHomeworkAverage(homework)
```

```
f  1# Compute and print the average of exam and homework scores
   2def printExamAverage(examScores):
   3    sum = 0
   4    for s in examScores:
   5        sum += s
   6    avg = sum / len(scores)
   7    print("Exam average:", avg)
   8
t  9def printHomeworkAverage(scores):
  10    sum = 0
  11    for s in scores:
  12        sum += s
  13    avg = sum / len(scores)
  14    print("Homework average:", avg)
  15
  16exams = [ 85, 90, 78 ]
  17homework = [ 100, 95, 88, 92 ]
  18
  19printExamAverage(exams)
  20printHomeworkAverage(homework)
```

Text events: 43-47

```
f  1# Compute and print the average of exam and homework scores
t  2def printExamAverage(examScores):
   3    sum = 0
   4    for s in examScores:
   5        sum += s
   6    avg = sum / len(scores)
   7    print("Exam average:", avg)
   8
   9def printHomeworkAverage(scores):
  10    sum = 0
  11    for s in scores:
  12        sum += s
  13    avg = sum / len(scores)
  14    print("Homework average:", avg)
  15
  16exams = [ 85, 90, 78 ]
  17homework = [ 100, 95, 88, 92 ]
  18
  19printExamAverage(exams)
  20printHomeworkAverage(homework)
```

```
f  1# Compute and print the average of exam and homework scores
t  2def printAverages(examScores):
   3    sum = 0
   4    for s in examScores:
   5        sum += s
   6    avg = sum / len(scores)
   7    print("Exam average:", avg)
   8
   9def printHomeworkAverage(scores):
  10    sum = 0
  11    for s in scores:
  12        sum += s
  13    avg = sum / len(scores)
  14    print("Homework average:", avg)
  15
  16exams = [ 85, 90, 78 ]
  17homework = [ 100, 95, 88, 92 ]
  18
  19printExamAverage(exams)
  20printHomeworkAverage(homework)
```

Text events: 48-65

```
# Compute and print the average of exam and homework scores
def printAverages(examScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

def printHomeworkAverage(scores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

def printHomeworkAverage(scores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

Text events: 66-76

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

def printHomeworkAverage(scores):
    sum = 0
    for s in scores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

def printHomeworkAverage(scores):
    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

Text events: 77-78

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

def printHomeworkAverage(scores):
    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)


    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

Text events: 79-83

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printExamAverage(exams)
printHomeworkAverage(homework)
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printAverages(exams)
printHomeworkAverage(homework)
```

Text events: 84-93

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printAverages(exams)
printHomeworkAverage(homework)
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printAverages(exams, homework)
printHomeworkAverage(homework)
```

Text events: 94-95

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printAverages(exams, homework)
printHomeworkAverage(homework)
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printAverages(exams, homework)
```

Text events: 108-109

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
exams = [ 85, 90, 78 ]
homework = [ 100, 95, 88, 92 ]

printAverages(exams, homework)
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)


```

Text events: 152-159

```
1# Compute and print the average of exam and homework scores
2def printAverages(examScores, homeworkScores):
3    sum = 0
4    for s in examScores:
5        sum += s
6    avg = sum / len(scores)
7    print("Exam average:", avg)
8
9    sum = 0
10   for s in homeworkScores:
11       sum += s
12   avg = sum / len(scores)
13   print("Homework average:", avg)
14
15def main():
16   exams = [ 85, 90, 78 ]
17   homework = [ 100, 95, 88, 92 ]
18
19   printAverages(exams, homework)
20
21if __name__ == "__main__":
```

```
1# Compute and print the average of exam and homework scores
2def printAverages(examScores, homeworkScores):
3    sum = 0
4    for s in examScores:
5        sum += s
6    avg = sum / len(scores)
7    print("Exam average:", avg)
8
9    sum = 0
10   for s in homeworkScores:
11       sum += s
12   avg = sum / len(scores)
13   print("Homework average:", avg)
14
15def main():
16   exams = [ 85, 90, 78 ]
17   homework = [ 100, 95, 88, 92 ]
18
19   printAverages(exams, homework)
20
21if __name__ == "__main__":
22   main()
23
```

Text events: 160-161

```
1# Compute and print the average of exam and homework scores
2def printAverages(examScores, homeworkScores):
3    sum = 0
4    for s in examScores:
5        sum += s
6    avg = sum / len(scores)
7    print("Exam average:", avg)
8
9    sum = 0
10   for s in homeworkScores:
11       sum += s
12   avg = sum / len(scores)
13   print("Homework average:", avg)
14
15def main():
16   exams = [ 85, 90, 78 ]
17   homework = [ 100, 95, 88, 92 ]
18
19   printAverages(exams, homework)
20
21if __name__ == "__main__":
22   main()
23
```

```
1# Compute and print the average of exam and homework scores
2def printAverages(examScores, homeworkScores):
3    sum = 0
4    for s in examScores:
5        sum += s
6    avg = sum / len(scores)
7    print("Exam average:", avg)
8
9    sum = 0
10   for s in homeworkScores:
11       sum += s
12   avg = sum / len(scores)
13   print("Homework average:", avg)
14
15def main():
16   exams = [ 85, 90, 78 ]
17   homework = [ 100, 95, 88, 92 ]
18
19   printAverages(exams, homework)
20
21if __name__ == "__main__":
22   main()
23
```

Text events: 162-167

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    avg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    examAvg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

Text events: 168-172

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    examAvg = sum / len(scores)
    print("Exam average:", avg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    examAvg = sum / len(scores)
    print("Exam average:", examAvg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

Text events: 173-177

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    examAvg = sum / len(scores)
    print("Exam average:", examAvg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()

```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    examAvg = sum / len(examScores)
    print("Exam average:", examAvg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()

```

Text events: 178-182

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    sum = 0
    for s in examScores:
        sum += s
    examAvg = sum / len(examScores)
    print("Exam average:", examAvg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()

```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        sum += s
    examAvg = sum / len(examScores)
    print("Exam average:", examAvg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()

```

Text events: 183-187

```
1# Compute and print the average of exam and homework scores
2def printAverages(examScores, homeworkScores):
3    examSum = 0
4    for s in examScores:
5        sum += s
6    examAvg = sum / len(examScores)
7    print("Exam average:", examAvg)
8
9    sum = 0
10   for s in homeworkScores:
11       sum += s
12   avg = sum / len(scores)
13   print("Homework average:", avg)
14
15def main():
16   exams = [ 85, 90, 78 ]
17   homework = [ 100, 95, 88, 92 ]
18
19   printAverages(exams, homework)
20
21if __name__ == "__main__":
22   main()
23
```

```
1# Compute and print the average of exam and homework scores
2def printAverages(examScores, homeworkScores):
3    examSum = 0
4    for s in examScores:
5        examSum += s
6    examAvg = sum / len(examScores)
7    print("Exam average:", examAvg)
8
9    sum = 0
10   for s in homeworkScores:
11       sum += s
12   avg = sum / len(scores)
13   print("Homework average:", avg)
14
15def main():
16   exams = [ 85, 90, 78 ]
17   homework = [ 100, 95, 88, 92 ]
18
19   printAverages(exams, homework)
20
21if __name__ == "__main__":
22   main()
23
```

Text events: 188-197

```
1# Compute and print the average of exam and homework scores
2def printAverages(examScores, homeworkScores):
3    examSum = 0
4    for s in examScores:
5        examSum += s
6    examAvg = sum / len(examScores)
7    print("Exam average:", examAvg)
8
9    sum = 0
10   for s in homeworkScores:
11       sum += s
12   avg = sum / len(scores)
13   print("Homework average:", avg)
14
15def main():
16   exams = [ 85, 90, 78 ]
17   homework = [ 100, 95, 88, 92 ]
18
19   printAverages(exams, homework)
20
21if __name__ == "__main__":
22   main()
23
```

```
1# Compute and print the average of exam and homework scores
2def printAverages(examScores, homeworkScores):
3    examSum = 0
4    for s in examScores:
5        examSum += s
6    examAvg = examSum / len(examScores)
7    print("Exam average:", examAvg)
8
9    sum = 0
10   for s in homeworkScores:
11       sum += s
12   avg = sum / len(scores)
13   print("Homework average:", avg)
14
15def main():
16   exams = [ 85, 90, 78 ]
17   homework = [ 100, 95, 88, 92 ]
18
19   printAverages(exams, homework)
20
21if __name__ == "__main__":
22   main()
23
```

Text events: 198-207

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        examSum += s
    examAvg = examSum / len(examScores)
    print("Exam average:", examAvg)

    sum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        examSum += s
    examAvg = examSum / len(examScores)
    print("Exam average:", examAvg)

    homeworkSum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

Text events: 208-216

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        examSum += s
    examAvg = examSum / len(examScores)
    print("Exam average:", examAvg)

    homeworkSum = 0
    for s in homeworkScores:
        sum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        examSum += s
    examAvg = examSum / len(examScores)
    print("Exam average:", examAvg)

    homeworkSum = 0
    for s in homeworkScores:
        homeworkSum += s
    avg = sum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

Text events: 232-240

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        examSum += s
    examAvg = examSum / len(examScores)
    print("Exam average:", examAvg)

    homeworkSum = 0
    for s in homeworkScores:
        homeworkSum += s
    avg = homeworkSum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        examSum += s
    examAvg = examSum / len(examScores)
    print("Exam average:", examAvg)

    homeworkSum = 0
    for s in homeworkScores:
        homeworkSum += s
    homeworkAvg = homeworkSum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

Text events: 241-249

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        examSum += s
    examAvg = examSum / len(examScores)
    print("Exam average:", examAvg)

    homeworkSum = 0
    for s in homeworkScores:
        homeworkSum += s
    homeworkAvg = homeworkSum / len(scores)
    print("Homework average:", avg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        examSum += s
    examAvg = examSum / len(examScores)
    print("Exam average:", examAvg)

    homeworkSum = 0
    for s in homeworkScores:
        homeworkSum += s
    homeworkAvg = homeworkSum / len(scores)
    print("Homework average:", homeworkAvg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

Text events: 250-258

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        examSum += s
    examAvg = examSum / len(examScores)
    print("Exam average:", examAvg)

    homeworkSum = 0
    for s in homeworkScores:
        homeworkSum += s
    homeworkAvg = homeworkSum / len(scores)
    print("Homework average:", homeworkAvg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

```
# Compute and print the average of exam and homework scores
def printAverages(examScores, homeworkScores):
    examSum = 0
    for s in examScores:
        examSum += s
    examAvg = examSum / len(examScores)
    print("Exam average:", examAvg)

    homeworkSum = 0
    for s in homeworkScores:
        homeworkSum += s
    homeworkAvg = homeworkSum / len(homeworkScores)
    print("Homework average:", homeworkAvg)

def main():
    exams = [ 85, 90, 78 ]
    homework = [ 100, 95, 88, 92 ]

    printAverages(exams, homework)

if __name__ == "__main__":
    main()
```

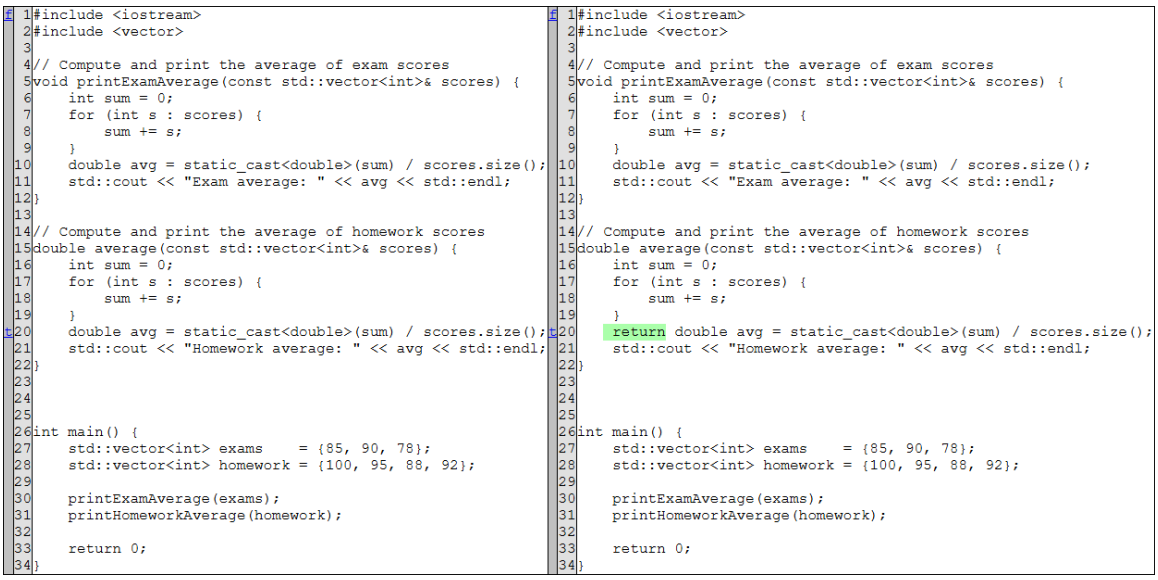
Task: generalizeAvgFunctions_2

Text events: 1-2

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
void printHomeworkAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
void printHomeworkAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}


int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

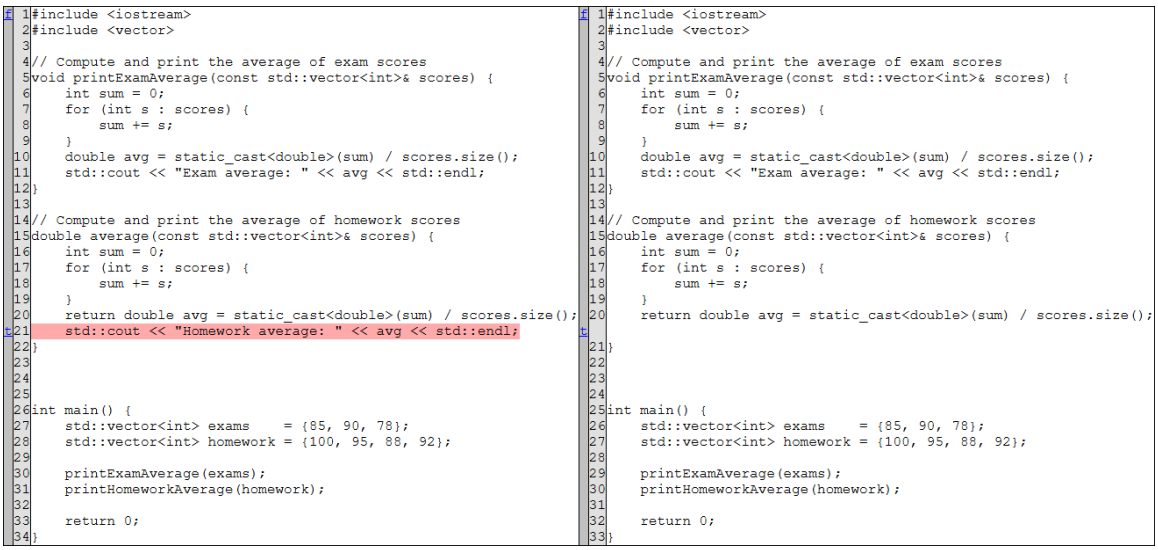
Text events: 32-38



```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 39-44



```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();

}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 51-52

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);

    printHomeworkAverage(homework);

    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
std::cout << "Homework average: " << avg << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

Text events:63-70

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);

    std::cout << "Homework average: " << avgerage << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);

    std::cout << "Homework average: " << avgerage(exames) << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 71-76

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);

    std::cout << "Homework average: " << avgerage(exames) << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};


    std::cout << "Homework average: " << avgerage(exames) << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 76-77

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};


    std::cout << "Homework average: " << avgerage(exames) << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};


    std::cout << "Homework average: " << avgerage(exams) << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 78-79

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};


    std::cout << "Homework average: " << avgerage(exams) << std::endl;

    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};


    std::cout << "Homework average: " << avgerage(exams) << std::endl;
std::cout << "Exam average: " << avg << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 95-98

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << avg << std::endl;
    std::cout << "Homework average: " << avgerage(exams) << std::endl;
    std::cout << "Exam average: " << avg << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average << std::endl;
    std::cout << "Homework average: " << avgerage(exams) << std::endl;
    std::cout << "Exam average: " << avg << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 99-107

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average << std::endl;
    std::cout << "Homework average: " << avgerage(exams) << std::endl;
    std::cout << "Exam average: " << avg << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << avgerage(exams) << std::endl;
    std::cout << "Exam average: " << avg << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 108-117

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << avgerage(exams) << std::endl;
    std::cout << "Exam average: " << avg << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << avgerage(homework) << std::endl;
    std::cout << "Exam average: " << avg << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 118-119

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << avgerage(homework) << std::endl;
    std::cout << "Exam average: " << avg << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << avgerage(homework) << std::endl;


    return 0;
}
```

Text events: 118-120

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << avgerage(homework) << std::endl;
    std::cout << "Exam average: " << avg << std::endl;
    printHomeworkAverage(homework);

    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << avgerage(homework) << std::endl;


    return 0;
}
```

Text events: 119-121

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << average(homework) << std::endl;

    printHomeworkAverage(homework);

    return 0;
}
```

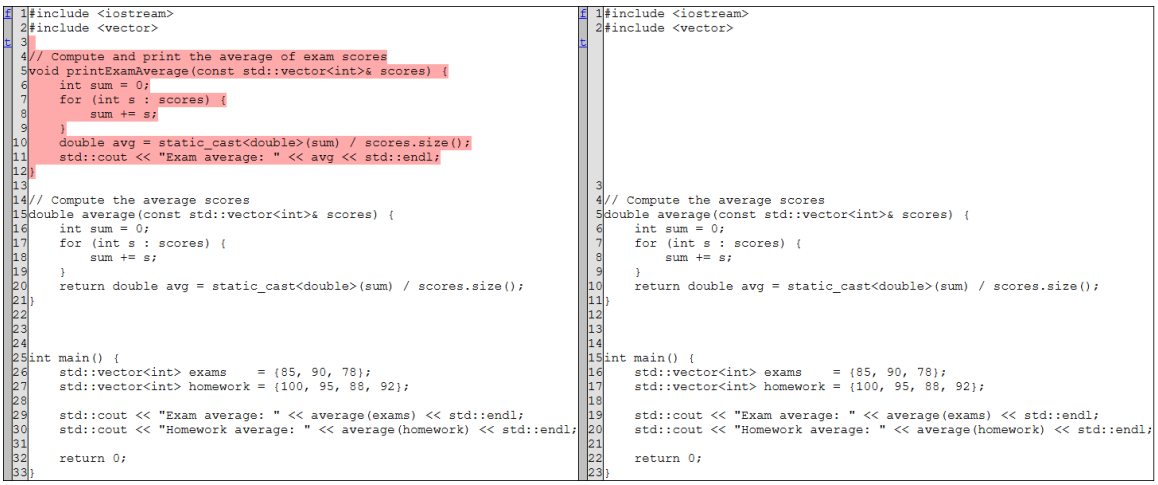
```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << average(homework) << std::endl;


    return 0;
}
```

Text events: 120-121

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << average(homework) << std::endl;


    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << average(homework) << std::endl;


    return 0;
}
```

Text events: 122-124

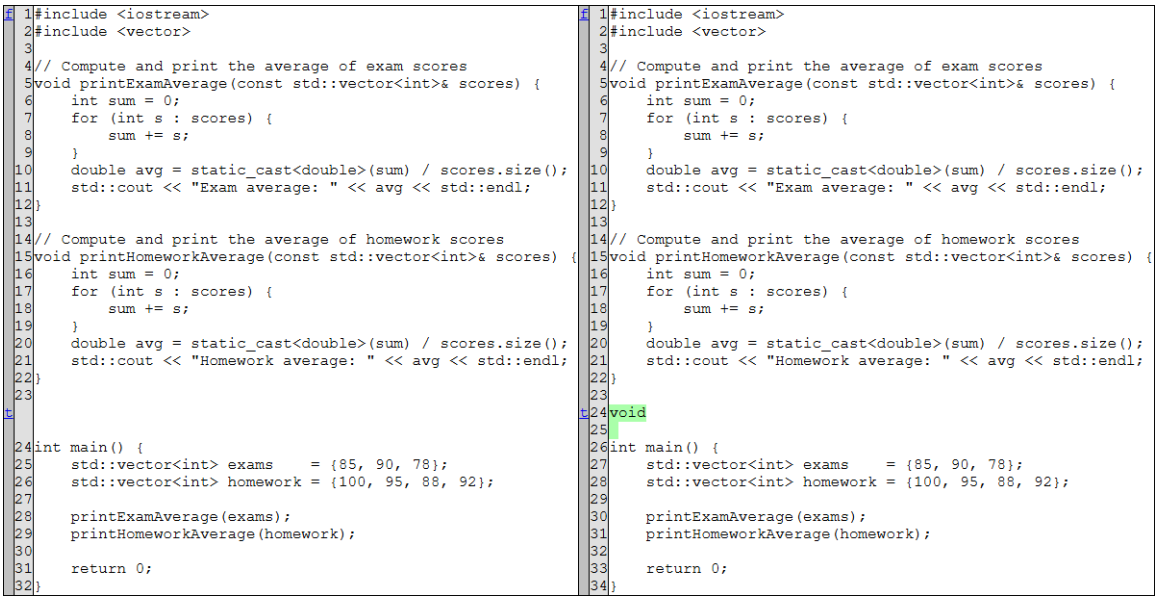


```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << average(homework) << std::endl;

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute the average scores
double average(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    return double avg = static_cast<double>(sum) / scores.size();
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    std::cout << "Exam average: " << average(exams) << std::endl;
    std::cout << "Homework average: " << average(homework) << std::endl;

    return 0;
}
```

Task: generalizeAvgFunctions_3

Text events: 1-6



```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
void printHomeworkAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}



int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
void printHomeworkAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}

void

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 58-58

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
void printHomeworkAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}

void printAverage(const std::vector<int>& scores){

}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
void printHomeworkAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}

void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 59-59

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
void printHomeworkAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}

void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
void printHomeworkAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}

void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 60-60

```
#include <iostream>
#include <vector>

// Compute and print the average of exam scores
void printExamAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Exam average: " << avg << std::endl;
}

// Compute and print the average of homework scores
void printHomeworkAverage(const std::vector<int>& scores) {
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Homework average: " << avg << std::endl;
}

void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 61-62

```
#include <iostream>
#include <vector>



void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

```
#include <iostream>
#include <vector>

// Compute and print the average of homework scores

void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printExamAverage(exams);
    printHomeworkAverage(homework);

    return 0;
}
```

Text events: 63-67

```
 1#include <iostream>
 2#include <vector>
 3
 4// Compute and print the average of homework scores
 5void printAverage(const std::vector<int>& scores){
 6    int sum = 0;
 7    for (int s : scores) {
 8        sum += s;
 9    }
10    double avg = static_cast<double>(sum) / scores.size();
11    std::cout << "Average: " << avg << std::endl;
12}
13
14int main() {
15    std::vector<int> exams    = {85, 90, 78};
16    std::vector<int> homework = {100, 95, 88, 92};
17
18    printExamAverage(exams);
19    printHomeworkAverage(homework);
20
21    return 0;
22}
```

```
 1#include <iostream>
 2#include <vector>
 3
 4// Compute and print the average of input scores
 5void printAverage(const std::vector<int>& scores){
 6    int sum = 0;
 7    for (int s : scores) {
 8        sum += s;
 9    }
10    double avg = static_cast<double>(sum) / scores.size();
11    std::cout << "Average: " << avg << std::endl;
12}
13
14int main() {
15    std::vector<int> exams    = {85, 90, 78};
16    std::vector<int> homework = {100, 95, 88, 92};
17
18    printExamAverage(exams);
19    printHomeworkAverage(homework);
20
21    return 0;
22}
```

Text events: 68-75

```
 1#include <iostream>
 2#include <vector>
 3
 4// Compute and print the average of input scores
 5void printAverage(const std::vector<int>& scores){
 6    int sum = 0;
 7    for (int s : scores) {
 8        sum += s;
 9    }
10    double avg = static_cast<double>(sum) / scores.size();
11    std::cout << "Average: " << avg << std::endl;
12}
13
14int main() {
15    std::vector<int> exams    = {85, 90, 78};
16    std::vector<int> homework = {100, 95, 88, 92};
17
18    printExamAverage(exams);
19    printHomeworkAverage(homework);
20
21    return 0;
22}
```

```
 1#include <iostream>
 2#include <vector>
 3
 4// Compute and print the average of input scores
 5void printAverage(const std::vector<int>& scores){
 6    int sum = 0;
 7    for (int s : scores) {
 8        sum += s;
 9    }
10    double avg = static_cast<double>(sum) / scores.size();
11    std::cout << "Average: " << avg << std::endl;
12}
13
14int main() {
15    std::vector<int> exams    = {85, 90, 78};
16    std::vector<int> homework = {100, 95, 88, 92};
17
18    printAverage(
19
20    return 0;
21}
```

Text events: 111-118

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of input scores
void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printAvave(exams

    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of input scores
void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printAverage(exams

    return 0;
}
```

Text events: 119-121

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of input scores
void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printAverage(exams


    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

// Compute and print the average of input scores
void printAverage(const std::vector<int>& scores){
    int sum = 0;
    for (int s : scores) {
        sum += s;
    }
    double avg = static_cast<double>(sum) / scores.size();
    std::cout << "Average: " << avg << std::endl;
}

int main() {
    std::vector<int> exams    = {85, 90, 78};
    std::vector<int> homework = {100, 95, 88, 92};

    printAverage(exams);


    return 0;
}
```

Task: implementFileSummation_1

Text events: 16-16

```
#include <iostream>
#include <fstream>

int readAndSumNumbers(std::ifstream& file) {
    int num1, num2;

}

int main() {
    std::ifstream numbers_file("data.txt");
    if (!numbers_file.is_open()) {
        exit(1);
    }

    while(!numbers_file.eof()) {
        readAndSumNumbers(numbers_file);
    }

    numbers_file.close()
}
```

```
#include <iostream>
#include <fstream>

int readAndSumNumbers(std::ifstream& file) {
    int num1, num2;

}

int main() {
    std::ifstream numbers_file("data.txt");
    if (!numbers_file.is_open()) {
        exit(1);
    }

    while(!numbers_file.eof()) {
        readAndSumNumbers(numbers_file);
    }

    numbers_file.close()
}
```

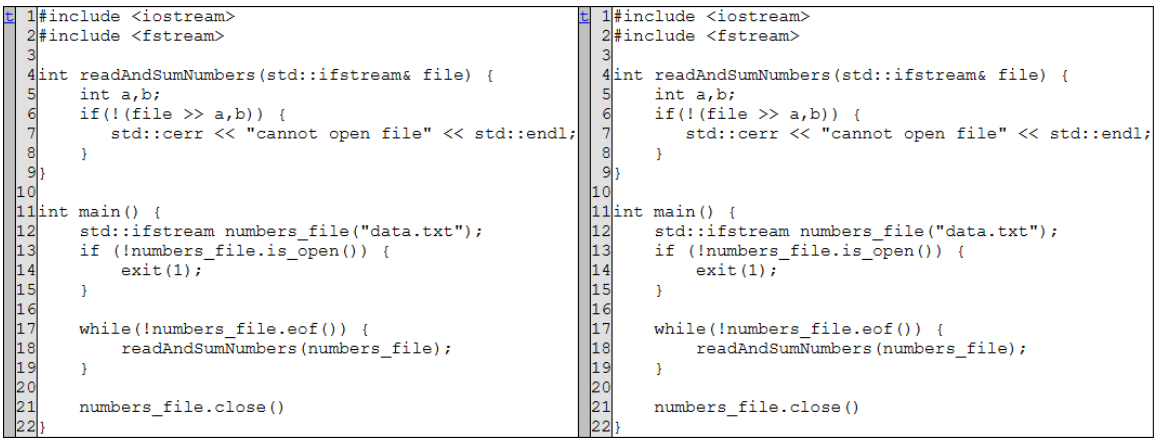
Text events: 101-105

```
#include <iostream>
#include <fstream>

int readAndSumNumbers(std::ifstream& file) {
    int num1, num2;
    file >> num1;
    file >> num2;

    std::cout << "Sum: "
    return num1 + num2;
}

int main() {
    std::ifstream numbers_file("data.txt");
    if (!numbers_file.is_open()) {
        exit(1);
    }

    while(!numbers_file.eof()) {
        readAndSumNumbers(numbers_file);
    }

    numbers_file.close()
}
```

```
#include <iostream>
#include <fstream>

int readAndSumNumbers(std::ifstream& file) {
    int num1, num2, sum;
    file >> num1;
    file >> num2;

    std::cout << "Sum: "
    return num1 + num2;
}

int main() {
    std::ifstream numbers_file("data.txt");
    if (!numbers_file.is_open()) {
        exit(1);
    }

    while(!numbers_file.eof()) {
        readAndSumNumbers(numbers_file);
    }

    numbers_file.close()
}
```

Task: implementFileSummation_2

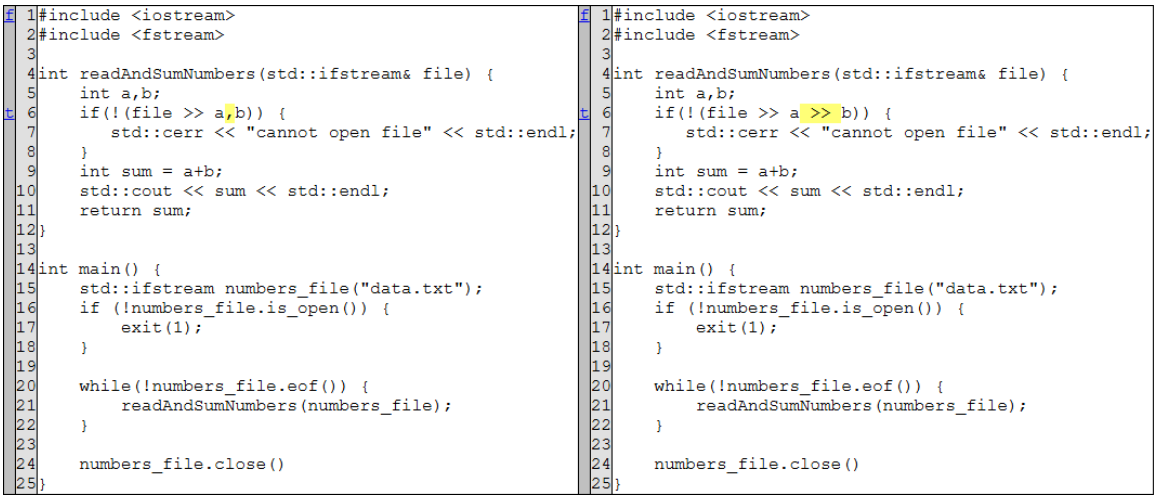
Text events: 107-108



```
#include <iostream>
#include <fstream>

int readAndSumNumbers(std::ifstream& file) {
    int a,b;
    if(!(file >> a,b)) {
       std::cerr << "cannot open file" << std::endl;
    }
}

int main() {
    std::ifstream numbers_file("data.txt");
    if (!numbers_file.is_open()) {
        exit(1);
    }

    while(!numbers_file.eof()) {
        readAndSumNumbers(numbers_file);
    }

    numbers_file.close()
}
```

```
#include <iostream>
#include <fstream>

int readAndSumNumbers(std::ifstream& file) {
    int a,b;
    if(!(file >> a,b)) {
       std::cerr << "cannot open file" << std::endl;
    }
}

int main() {
    std::ifstream numbers_file("data.txt");
    if (!numbers_file.is_open()) {
        exit(1);
    }

    while(!numbers_file.eof()) {
        readAndSumNumbers(numbers_file);
    }

    numbers_file.close()
}
```

Text events: 181-185



```
#include <iostream>
#include <fstream>

int readAndSumNumbers(std::ifstream& file) {
    int a,b;
    if(!(file >> a,b)) {
       std::cerr << "cannot open file" << std::endl;
    }
    int sum = a+b;
    std::cout << sum << std::endl;
    return sum;
}

int main() {
    std::ifstream numbers_file("data.txt");
    if (!numbers_file.is_open()) {
        exit(1);
    }

    while(!numbers_file.eof()) {
        readAndSumNumbers(numbers_file);
    }

    numbers_file.close()
}
```

```
#include <iostream>
#include <fstream>

int readAndSumNumbers(std::ifstream& file) {
    int a,b;
    if(!(file >> a >> b)) {
       std::cerr << "cannot open file" << std::endl;
    }
    int sum = a+b;
    std::cout << sum << std::endl;
    return sum;
}

int main() {
    std::ifstream numbers_file("data.txt");
    if (!numbers_file.is_open()) {
        exit(1);
    }

    while(!numbers_file.eof()) {
        readAndSumNumbers(numbers_file);
    }

    numbers_file.close()
}
```

Task: implementFileSummation_3

Text events: 1-4

```
def readAndSumNumbers(line):
    pass

numbers_file = open("data.txt",'r')

for line in numbers_file:
    readAndSumNumbers(line)

numbers_file.close()
```

```
def readAndSumNumbers(line):

numbers_file = open("data.txt",'r')

for line in numbers_file:
    readAndSumNumbers(line)

numbers_file.close()
```

Text events: 50-55

```
def readAndSumNumbers(line):
    x, y = line.split(" ")
    print(x, y)


numbers_file = open("data.txt",'r')

for line in numbers_file:
    readAndSumNumbers(line)

numbers_file.close()
```

```
def readAndSumNumbers(line):
    x, y = line.split(" ")
    print(x + y)


numbers_file = open("data.txt",'r')

for line in numbers_file:
    readAndSumNumbers(line)

numbers_file.close()
```

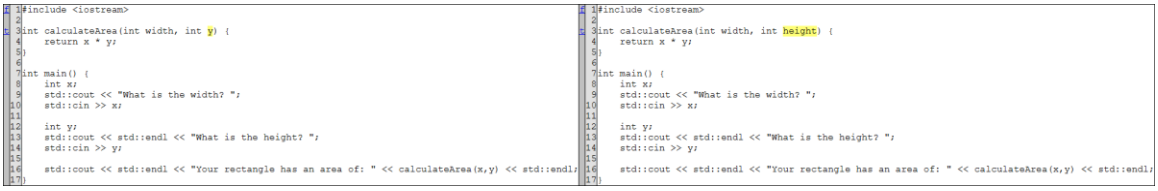
Task: nameChange_1

Text events: 1-6

```
#include <iostream>

int calculateArea(int x, int y) {
    return x * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int y) {
    return x * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

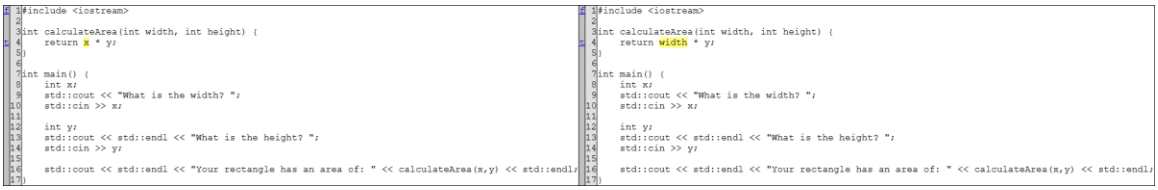
Text events: 7-13



```
#include <iostream>

int calculateArea(int width, int y) {
    return x * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return x * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

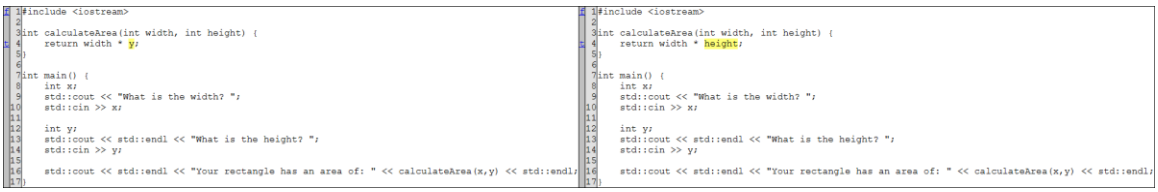
Text events: 14-19



```
#include <iostream>

int calculateArea(int width, int height) {
    return x * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

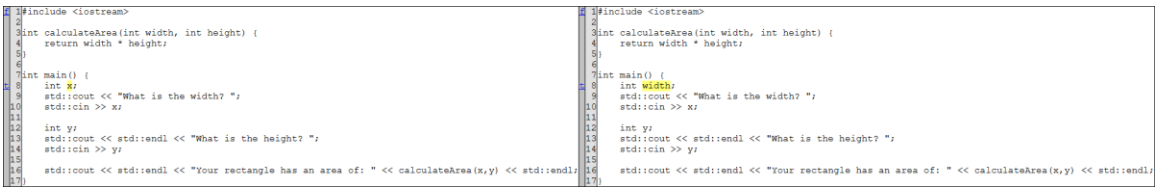
Text events: 20-26



```
#include <iostream>

int calculateArea(int width, int height) {
    return width * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

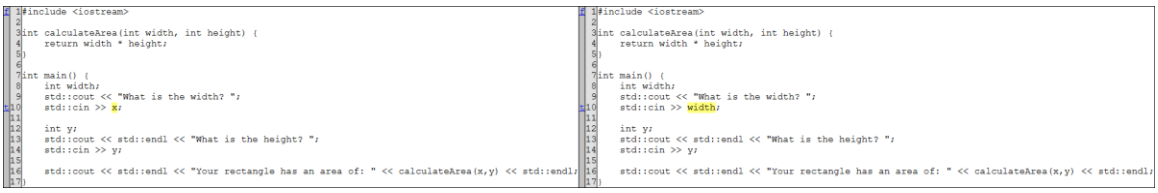
Text events: 27-32



```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

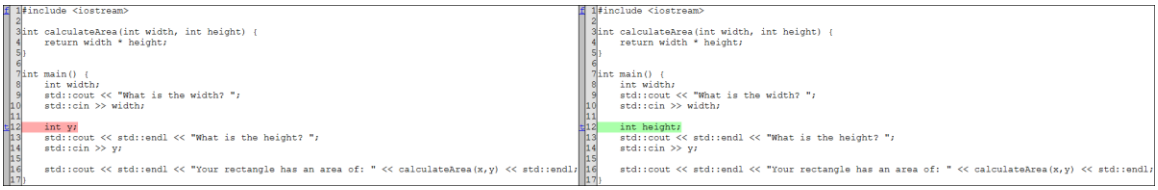
Text events: 33-38



```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

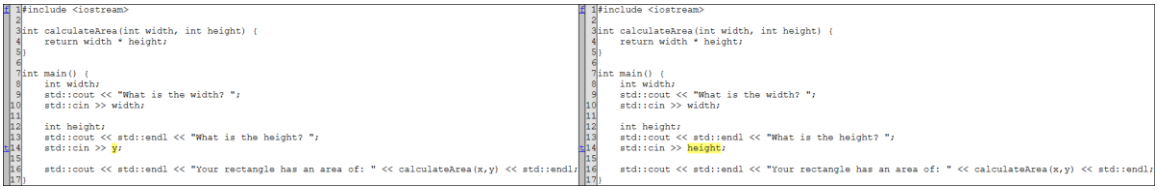
Text events: 39-45



```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

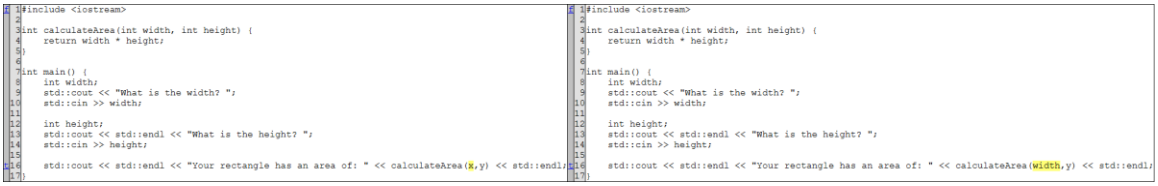
Text events: 46-52



```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> height;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

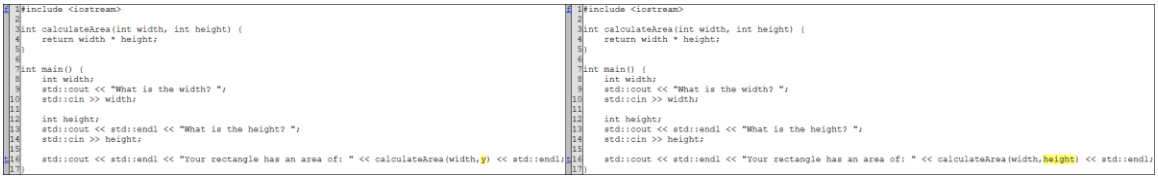
Text events: 53-58



```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> height;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> height;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

Text events: 59-67

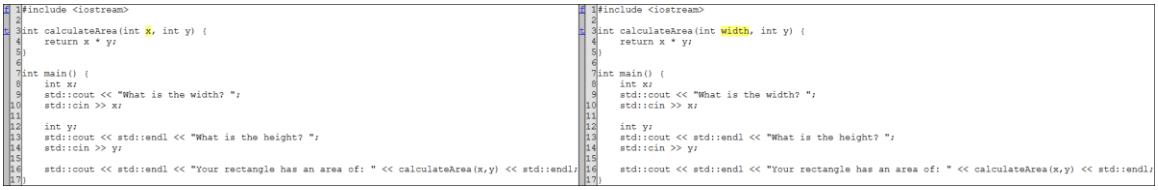


```cpp
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> height;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

```cpp
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> height;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,height) << std::endl;
}
```

Task: nameChange_2

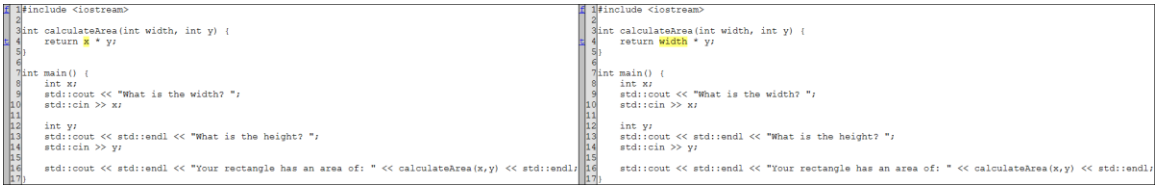
Text events: 1-6



```cpp
#include <iostream>

int calculateArea(int x, int y) {
    return x * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```cpp
#include <iostream>

int calculateArea(int width, int y) {
    return x * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

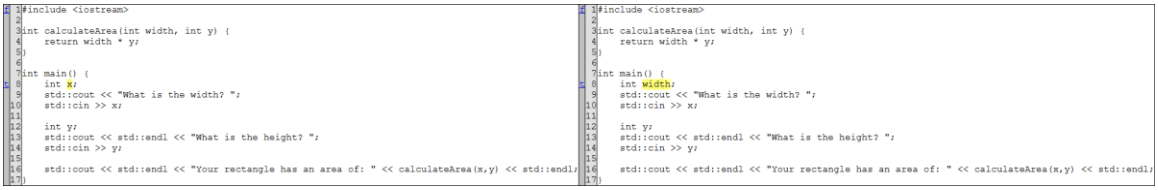
Text events: 7-8



```cpp
#include <iostream>

int calculateArea(int width, int y) {
    return x * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```cpp
#include <iostream>

int calculateArea(int width, int y) {
    return width * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

Text events: 9-10



```cpp
#include <iostream>

int calculateArea(int width, int y) {
    return width * y;
}

int main() {
    int x;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```cpp
#include <iostream>

int calculateArea(int width, int y) {
    return width * y;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

Text events: 11-12

```
#include <iostream>

int calculateArea(int width, int y) {
    return width * y;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> x;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int y) {
    return width * y;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

Text events: 13-14

```
#include <iostream>

int calculateArea(int width, int y) {
    return width * y;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(x,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int y) {
    return width * y;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

Text events: 15-21

```
#include <iostream>

int calculateArea(int width, int y) {
    return width * y;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * y;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

Text events: 22-23

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * y;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

Text events: 24-25

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int y;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

Text events: 26-27

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> y;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> height;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

Text events: 28-29

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> height;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> height;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

Text events: 30-31

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> height;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,y) << std::endl;
}
```

```
#include <iostream>

int calculateArea(int width, int height) {
    return width * height;
}

int main() {
    int width;
    std::cout << "What is the width? ";
    std::cin >> width;

    int height;
    std::cout << std::endl << "What is the height? ";
    std::cin >> height;

    std::cout << std::endl << "Your rectangle has an area of: " << calculateArea(width,height) << std::endl;
}
```

Task: nameChange_3

Text events: 1-6

```
def calculateArea(x, y):
    return x * y

x = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(x,y))
```

```
def calculateArea(width, y):
    return x * y

x = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(x,y))
```

Text events: 7-8

```
def calculateArea(width, y):
    return x * y

x = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(x,y))
```

```
def calculateArea(width, y):
    return width * y

x = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(x,y))
```

Text events: 9-10

```
def calculateArea(width, y):
    return width * y

x = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(x,y))
```

```
def calculateArea(width, y):
    return width * y

width = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(x,y))
```

Text events: 11-12

```
def calculateArea(width, y):
    return width * y

width = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(x,y))
```

```
def calculateArea(width, y):
    return width * y

width = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(width,y))
```

Text events: 13-25

```
def calculateArea(width, y):
    return width * y

width = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(width,y))
```

```
def calculateArea(width, height):
    return width * y

width = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(width,y))
```

Text events: 26-27

```
def calculateArea(width, height):
    return width * y

width = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(width,y))
```

```
def calculateArea(width, height):
    return width * height

width = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(width,y))
```

Text events: 28-29

```
def calculateArea(width, height):
    return width * height

width = int(input("What is the width? "))
y = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(width,y))
```

```
def calculateArea(width, height):
    return width * height

width = int(input("What is the width? "))
height = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(width,y))
```

Text events: 30-31

```
def calculateArea(width, height):
    return width * height

width = int(input("What is the width? "))
height = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(width,y))
```

```
def calculateArea(width, height):
    return width * height

width = int(input("What is the width? "))
height = int(input("What is the height? "))
print("Your rectangle has an area of:",calculateArea(width,height))
```

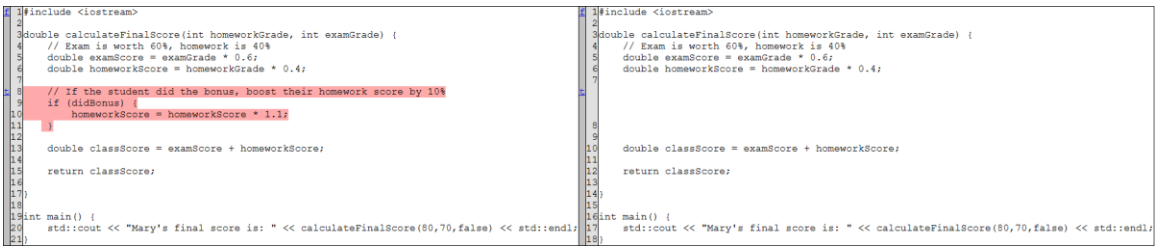
Task: removeDeadCode_1

Text events: 1-1

```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade, bool didBonus) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;

    // If the student did the bonus, boost their homework score by 10%
    if (didBonus) {
        homeworkScore = homeworkScore * 1.1;
    }

    double classScore = examScore + homeworkScore;

    return classScore;
}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;

    // If the student did the bonus, boost their homework score by 10%
    if (didBonus) {
        homeworkScore = homeworkScore * 1.1;
    }

    double classScore = examScore + homeworkScore;

    return classScore;
}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

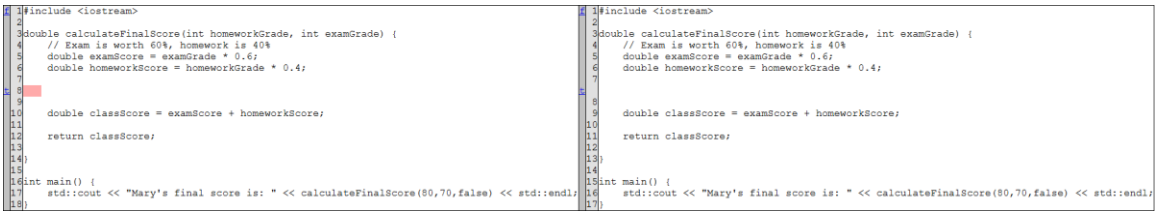
Text events: 2-2



```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;

    // If the student did the bonus, boost their homework score by 10%
    if (didBonus) {
        homeworkScore = homeworkScore * 1.1;
    }

    double classScore = examScore + homeworkScore;

    return classScore;

}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;



    double classScore = examScore + homeworkScore;

    return classScore;

}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

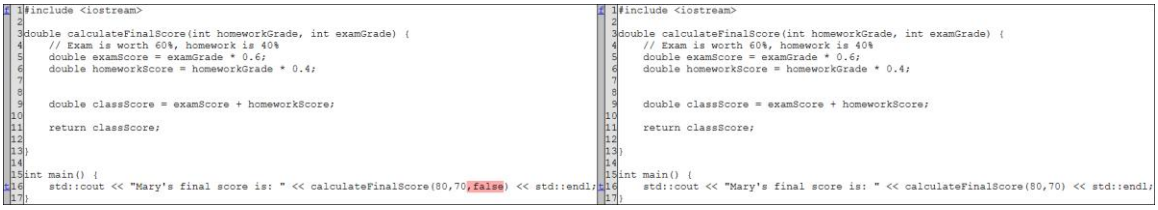
Text events: 3-6



```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;


    double classScore = examScore + homeworkScore;

    return classScore;

}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;


    double classScore = examScore + homeworkScore;

    return classScore;

}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

Text events: 7-7

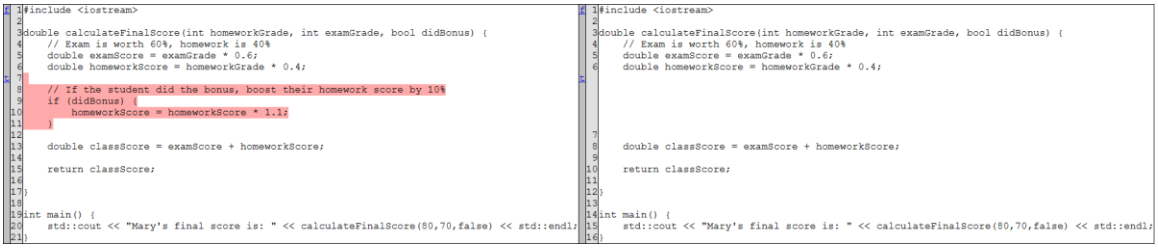


```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;


    double classScore = examScore + homeworkScore;

    return classScore;

}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;


    double classScore = examScore + homeworkScore;

    return classScore;

}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70) << std::endl;
}
```

Task: removeDeadCode_2

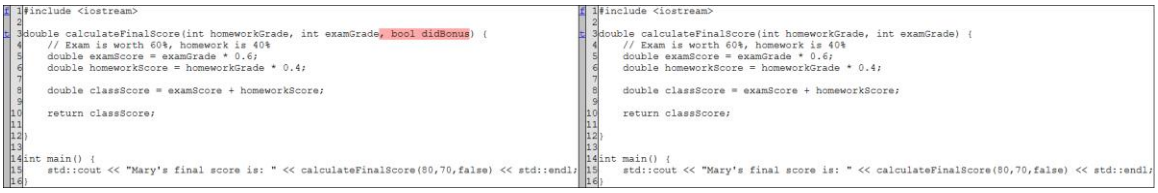
Text events: 1-3



```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade, bool didBonus) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;

    // If the student did the bonus, boost their homework score by 10%
    if (didBonus) {
        homeworkScore = homeworkScore * 1.1;
    }

    double classScore = examScore + homeworkScore;

    return classScore;
}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade, bool didBonus) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;

    double classScore = examScore + homeworkScore;

    return classScore;
}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

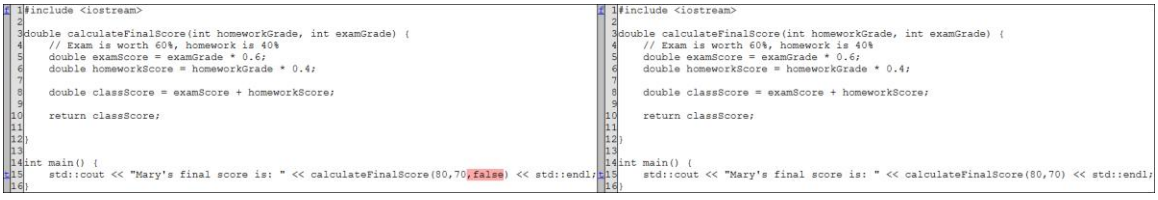
Text events:



```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade, bool didBonus) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;

    double classScore = examScore + homeworkScore;

    return classScore;
}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;

    double classScore = examScore + homeworkScore;

    return classScore;
}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

Text events: 5-5

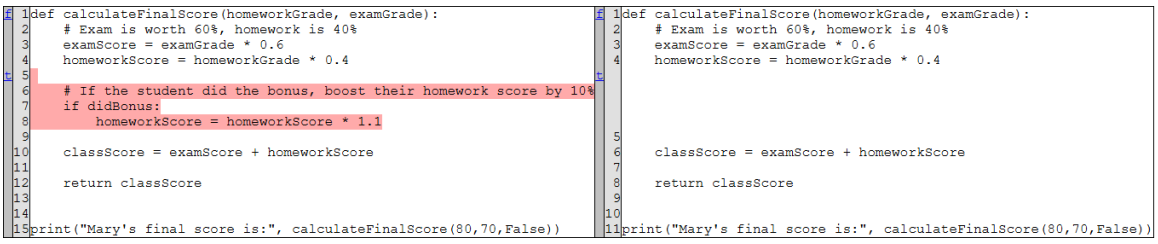


```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;

    double classScore = examScore + homeworkScore;

    return classScore;
}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70,false) << std::endl;
}
```

```
#include <iostream>

double calculateFinalScore(int homeworkGrade, int examGrade) {
    // Exam is worth 60%, homework is 40%
    double examScore = examGrade * 0.6;
    double homeworkScore = homeworkGrade * 0.4;

    double classScore = examScore + homeworkScore;

    return classScore;
}

int main() {
    std::cout << "Mary's final score is: " << calculateFinalScore(80,70) << std::endl;
}
```

Task: removeDeadCode_3

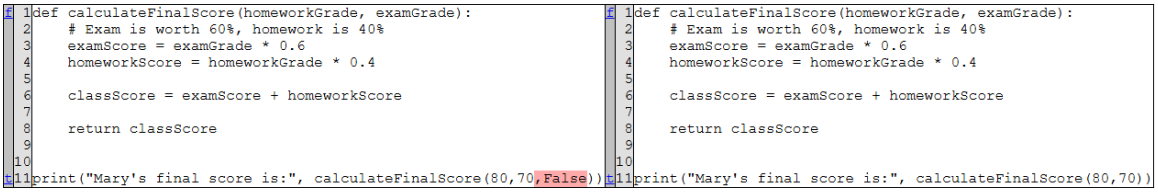
Text events: 11-16



```
def calculateFinalScore(homeworkGrade, examGrade):
    # Exam is worth 60%, homework is 40%
    examScore = examGrade * 0.6
    homeworkScore = homeworkGrade * 0.4

    # If the student did the bonus, boost their homework score by 10%
    if didBonus:
        homeworkScore = homeworkScore * 1.1

    classScore = examScore + homeworkScore

    return classScore


print("Mary's final score is:", calculateFinalScore(80,70,False))
```

```
def calculateFinalScore(homeworkGrade, examGrade):
    # Exam is worth 60%, homework is 40%
    examScore = examGrade * 0.6
    homeworkScore = homeworkGrade * 0.4


    classScore = examScore + homeworkScore

    return classScore


print("Mary's final score is:", calculateFinalScore(80,70,False))
```

Text events: 17-17

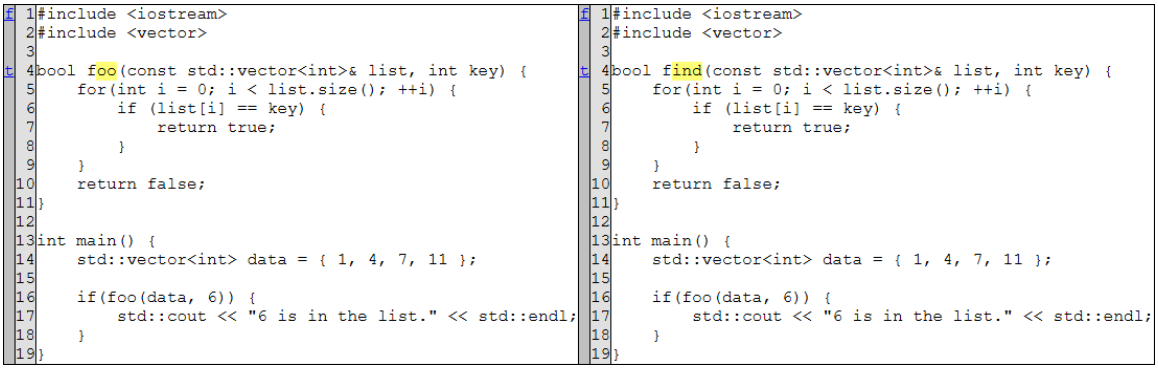


```
def calculateFinalScore(homeworkGrade, examGrade):
    # Exam is worth 60%, homework is 40%
    examScore = examGrade * 0.6
    homeworkScore = homeworkGrade * 0.4

    classScore = examScore + homeworkScore

    return classScore


print("Mary's final score is:", calculateFinalScore(80,70,False))
```

```
def calculateFinalScore(homeworkGrade, examGrade):
    # Exam is worth 60%, homework is 40%
    examScore = examGrade * 0.6
    homeworkScore = homeworkGrade * 0.4

    classScore = examScore + homeworkScore

    return classScore


print("Mary's final score is:", calculateFinalScore(80,70))
```

Task: renameFunction_1

Text events: 1-4



```
#include <iostream>
#include <vector>

bool foo(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

```
#include <iostream>
#include <vector>

bool find(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

Text events: 5-14

```
#include <iostream>
#include <vector>

bool find(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

```
#include <iostream>
#include <vector>

bool search(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

Text events: 15-24

```
#include <iostream>
#include <vector>

bool search(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

```
#include <iostream>
#include <vector>

bool find(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

Text events: 25-30

```
#include <iostream>
#include <vector>

bool find(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

```
#include <iostream>
#include <vector>

bool search(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

Text events: 31-36

```
#include <iostream>
#include <vector>

bool search(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

```
#include <iostream>
#include <vector>

bool search(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(search(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

Task: renameFunction_2

Text events: 1-26

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean foo(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (foo(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean findNumberOcurrences(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (foo(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

Text events: 27-40

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean findNumberOcurrences(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (foo(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean isNumberInList(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (foo(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

Text events: 41-58

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean isNumberInList(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (foo(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean linearNumberSearch(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (foo(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

Text events: 59-59

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean linearNumberSearch(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (foo(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean linearNumberSearch(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (linearNumberSearch(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

Text events: 60-75

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean linearNumberSearch(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (linearNumberSearch(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean isNumberInList(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (linearNumberSearch(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

Text events: 76-89

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean isNumberInList(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (linearNumberSearch(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Main {
    static boolean isNumberInList(List<Integer> list, int key) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) == key) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        List<Integer> data = Arrays.asList(1, 4, 7, 11);

        if (isNumberInList(data, 6)) {
            System.out.println("6 is in the list.");
        }
    }
}
```

Task: renameFunction_3

Text events: 1-3

```cpp
#include <iostream>
#include <vector>

bool foo(const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

```cpp
#include <iostream>
#include <vector>

bool (const std::vector<int>& list, int key) {
    for(int i = 0; i < list.size(); ++i) {
        if (list[i] == key) {
            return true;
        }
    }
    return false;
}

int main() {
    std::vector<int> data = { 1, 4, 7, 11 };

    if(foo(data, 6)) {
        std::cout << "6 is in the list." << std::endl;
    }
}
```

Text events: 4-11

```
f  1#include <iostream>
   2#include <vector>
   3
t  4bool (const std::vector<int>& list, int key) {
   5    for(int i = 0; i < list.size(); ++i) {
   6        if (list[i] == key) {
   7            return true;
   8        }
   9    }
  10    return false;
  11}
  12
  13int main() {
  14    std::vector<int> data = { 1, 4, 7, 11 };
  15
  16    if(foo(data, 6)) {
  17        std::cout << "6 is in the list." << std::endl;
  18    }
  19}
```

```
f  1#include <iostream>
   2#include <vector>
   3
t  4bool contains(const std::vector<int>& list, int key) {
   5    for(int i = 0; i < list.size(); ++i) {
   6        if (list[i] == key) {
   7            return true;
   8        }
   9    }
  10    return false;
  11}
  12
  13int main() {
  14    std::vector<int> data = { 1, 4, 7, 11 };
  15
  16    if(foo(data, 6)) {
  17        std::cout << "6 is in the list." << std::endl;
  18    }
  19}
```

APPENDIX C

Edit Detection Algorithm Pseudo Code Implementations

Simple Algorithm

```
PROCEDURE SimpleAlgorithm:
    INPUT: TEXT_EVENT[] text_events
    OUTPUT: EDIT[]

    EDIT[] edits

    FOR text_event IN text_events:
        EDIT edit = create_edit({text_event})
        edits.append(edit)
    ENDFOR

    RETURN edits
END PROCEDURE
```

Temporal Algorithm

```
PROCEDURE TemporalAlgorithm:
    INPUT: TEXT_EVENT[] text_events
           NUM gap_in_ms
    OUTPUT: EDIT[]

    EDIT[] edits
    TEXT_EVENT[] event_vec

    FOR text_event IN text_events:
        IF event_vec.len != 0 AND
           text_event.timestamp – event_vec.last.timestamp > gap_in_ms

            EDIT edit = create_edit(event_vec)
            edits.append(edit)
            event_vec.clear()
        ENDIF
        event_vec.append(text_event)
    ENDFOR

    RETURN edits
END PROCEDURE
```

Dynamic Temporal Algorithm

```
PROCEDURE DynamicTemporalAlgorithm:
    INPUT: TEXT_EVENT[] text_events
           NUM scalar
    OUTPUT: EDIT[]

    EDIT[] edits
    TEXT_EVENT[] event_vec

    NUM length = text_events.last.timestamp – text_events.first.timestamp
    NUM gap_in_ms = length / text_events.len * scalar

    FOR text_event IN text_events:
        IF event_vec.len != 0 AND
           text_event.timestamp – event_vec.last.timestamp > gap_in_ms

            EDIT edit = create_edit(event_vec)
            edits.append(edit)
            event_vec.clear()
        ENDIF
        event_vec.append(text_event)
    ENDFOR

    RETURN edits
END PROCEDURE
```

Token Algorithm

```
PROCEDURE TokenAlgorithm:
    INPUT: TEXT_EVENT[] text_events
    OUTPUT: EDIT[]

    EDIT[] edits
    TEXT_EVENT[] event_vec
    TOKEN last_edited_token

    FOR text_event IN text_events:
        TOKEN[] prev_tokens = tokenize(event_vec)
        TOKEN[] crnt_tokens = tokenize(event_vec+{text_event})


        TOKEN changed_token
        WHILE prev_tokens[i] != crnt_tokens[i]:
            changed_token = crnt_token[i]
            ++i
        ENDWHILE

        IF changed_token != last_edited_token
            EDIT edit = create_edit(event_vec)
            edits.append(edit)
            event_vec.clear()
        ENDIF


        event_vec.append(text_event)
        last_edited_token = changed_token

    ENDFOR

    RETURN edits
END PROCEDURE
```

Heuristics Algorithm

```
PROCEDURE HeuristicsAlgorithm:
    INPUT: TEXT_EVENT[] text_events
           NUM gap_in_ms
    OUTPUT: EDIT[]

    EDIT[] edits
    TEXT_EVENT[] event_vec
    TEXT_EVENT prev

    FOR text_event IN text_events:
        IF text_event.timestamp – event_vec.last.timestamp > gap_in_ms OR
           (event.inserted.contains("\n") AND isNotAllWs(event_vec)) OR
           (isNotAllWs({text_event,prev}) AND event.line != prev.line)

            EDIT edit = create_edit(event_vec)
            edits.append(edit)
            event_vec.clear()
        ENDIF
        event_vec.append(text_event)
        last = text_event
    ENDFOR

    RETURN edits
END PROCEDURE
```

Heuristics+Dynamic Temporal Algorithm

```
PROCEDURE HeuristicsDynamicTemporalAlgorithm:
    INPUT: TEXT_EVENT[] text_events
           NUM scalar
    OUTPUT: EDIT[]

    EDIT[] edits
    TEXT_EVENT[] event_vec
    TEXT_EVENT prev

    NUM length = text_events.last.timestamp – text_events.first.timestamp
    NUM gap_in_ms = length / text_events.len * scalar


    FOR text_event IN text_events:
        IF text_event.timestamp – event_vec.last.timestamp > gap_in_ms OR
           (event.inserted.contains("\n") AND isNotAllWs(event_vec)) OR
           (isNotAllWs({text_event,prev}) AND event.line != prev.line)

            EDIT edit = create_edit(event_vec)
            edits.append(edit)
            event_vec.clear()
        ENDIF
        event_vec.append(text_event)
        last = text_event
    ENDFOR

    RETURN edits
END PROCEDURE
```