Dissertation on

"Voice Assistant for Programming"

Submitted in partial fulfillment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Himanshu Sonthalia	**01FB15ECS126**
Varun V	**01FB15ECS341**

Under the guidance of

Internal Guide
Mr. Channa Bankapur
Assistant Professor,
PES University

January – May 2019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

'Voice Assistant for Programming'

is a bonafide work carried out by

Himanshu Sonthalia	**01FB15ECS126**
Varun V	**01FB15ECS341**

In partial fulfilment for the completion of eighth semester project work in the Program of Study Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2019 – May. 2019. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th semester academic requirements in respect of project work.

Signature Mr. Channa Bankapur Assistant Professor	Signature Dr. Shylaja S S Chairperson	Signature Dr. B K Keshavan Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1. _____	_____
2. _____	_____

DECLARATION

We hereby declare that the project entitled **"Voice Assistant for Programming"** has been carried out by us under the guidance of Prof. Channa Bankapur and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester January – May 2019. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

01FB15ECS126	**Himanshu Sonthalia**	_____
01FB15ECS341	**Varun V**	_____

ACKNOWLEDGEMENT

I would like to express my gratitude to Prof. Channa Bankapur, Dept. of Computer Science, PES University, for his continuous guidance, assistance and encouragement throughout the development of this project.

I am grateful to the project coordinators, Prof. Preet Kanwal, Prof. Sangeetha for organizing, managing and helping out with the entire process.

I take this opportunity to thank Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department. I would also like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help.

I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. K.N.B. Murthy, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement I have received from my mother.

ABSTRACT

The goal of this project is to develop a Voice Assistant for programming in C programming language. It shall be implemented as a Visual Studio Code extension which will provide the interface and basic features and commands to work with. The assistant lets the user write C programs, using the constructs of C, by simply giving Natural Language commands. Through voice commands the user can navigate across the file and edit the code with intuitive natural language statements.

When the voice input is detected, a response text is generated by Google Cloud API, it is sent to the parser. The parser will parse the input and generate its corresponding semantics. Next, the executor shall try to execute the command to generate the code, catch errors, perform actions such as navigate, edit, start dictation, etc.

Table of Content

List of Figures

CHAPTER - 1

INTRODUCTION

Today coding through vocal inputs is not very popular or sometimes even feasible simply due to lack of work that's been done in this area and products available for it. Programmers throughout the world suffer from repetitive strain injury, carpal tunnel syndrome, etc. due to always being bound to a computer. There are a huge group of people who are physically challenged and cannot type but have tremendous knowledge and skills in coding. Few software which exist allowing users to code vocally, expect the user to adapt to them and their completely new language, grammar, etc.

We all know that our thoughts are more natural language oriented. Thinking natural language commands/ statements is highly intuitive and much easier. We then convert it to the coding language we are trying to code in. Coding using just natural language imperative statements and getting it in the desired language would not only make coding faster, easier and neater but also reduce silly mistakes, syntactical errors and all manual works in general. Yet very little developments have been made in its direction.

IDE's these days help not only to code and set up and maintain user's environment or other basic tasks but provides a graphical user interface (GUI)-based workbench designed to aid a developer in building software applications, extensions, contribute code, etc with an integrated environment combined with all the required tools at hand. Quite a few IDE's are being developed hoping to improve programming experience, by simplifying, automating, etc., however, voice-based IDE's are rare and provide little or no assistance for vocal features.

In this project, a Visual Studio Code extension is presented, which will provide an Interactive Voice Assistant for Programming in C language. Users will give vocal commands that too in natural language. It will translate the high-level Natural Language statements given as input, to C code. It shall also provide additional features such as navigation, editing, etc.

We implement a simple version having support for most of the popular C language constructs. We try to achieve a simple useable model which can be extended and ported as required. Through voice commands the user can navigate across the file and edit the code using simple natural language statements. The errors will be neatly reported, alternative suggestions provided to account for both lexical ambiguities and syntactic ambiguities.

The overall flow of our system and the work being done on the input may be described as follows :

- When the user gives a command, it will be received, processed and converted to text by Google's Cloud Speech to Text API. It will send a list of possible strings for given voice input. Using confidence and scores of these strings, we select the best response and send it for parsing.

- The parser will try to parse the string to see if the input string is a part of the language and is supported by the grammar and rules defined. On successful parse, a parse tree is created for the input, and its corresponding semantic is generated.

- The executor takes the semantics of the given input and tries to make sense and execute the command. The command may be a C language construct which needs to be inserted in the editor or a general command to the assistant, making no change in the code, but telling it to perform tasks or operations such as, "go to line 3", "undo", "redo", etc.

- VS Code being our front end handles all outputs whether they are suggestions, errors, dictation output, etc. Errors encountered across the system will be properly collected and passed preventing system crashes and failures. The error message shall be reported as voice feedback and also logged in the errors pane in our panel. The corresponding input which caused the error shall be displayed in the suggestions pane and other information about the error type, state of the input and how much has been translated shall be displayed if available.

CHAPTER - 2

PROBLEM DEFINITION

Today there is an increase in the general trend for coding. Whether the goal is for pure apps and software developmental purposes, or for writing simple programs and macros for non-CS background people. Yet very little assistance within a language is provided for coding. The features being given by IDE's today are more of aesthetic, management, etc, type to the environment.

Always having to type out codes, as programmers we are restricted to a keyboard and screen. However, some of the situations where hands-free coding may be truly beneficial is :

- Programmers suffering from repetitive strain injury, carpal tunnel syndrome, etc. or having physical disabilities, hence, unable to type.
- Situations where it's inconvenient to type, eg. on mobile devices, or while on the go, or while teaching, etc.

Our thoughts are more natural language oriented. Vocal control and natural language commands are more intuitive and easier. Being restricted in terms of language (in contrast with using natural language) requires us to be very well versed with the syntaxes and watch out for the popular mistakes, for all programmers, whether new to a language or using multiple often causing trouble remembering the syntax. Repetitive and standard constructs are being written often, resulting in silly and redundant mistakes, eg: missing brackets, semi-colons, etc

CHAPTER - 3

LITERATURE SURVEY

A literature survey provides a description, summary, and critical evaluation of the works in relation to the research problem being investigated. It guides or helps the researcher to define/find out/identify a problem and see all available solutions and works done or being done. For this project, we have performed an exhaustive literature survey as not many exist. The few papers which exist are old and has slowly been forgotten. Among the available papers and works done, most of the related works focus on using a fixed language for programming. These languages are non- intuitive and highly restrictive.

The most relevant papers which have actually been helpful in defining the problems faced, approaches, ambiguities, etc are listed as below :

3.1 NaturalJava: A Natural Language Interface for Programming in Java

3.1.1 Introduction

Natural Java [1] is a natural language interface to the Java programming language. It has three components

- Sundance - a natural language processor,
- PRISM - a decision tree based case frame interpreter
- TreeFace - an AST manager

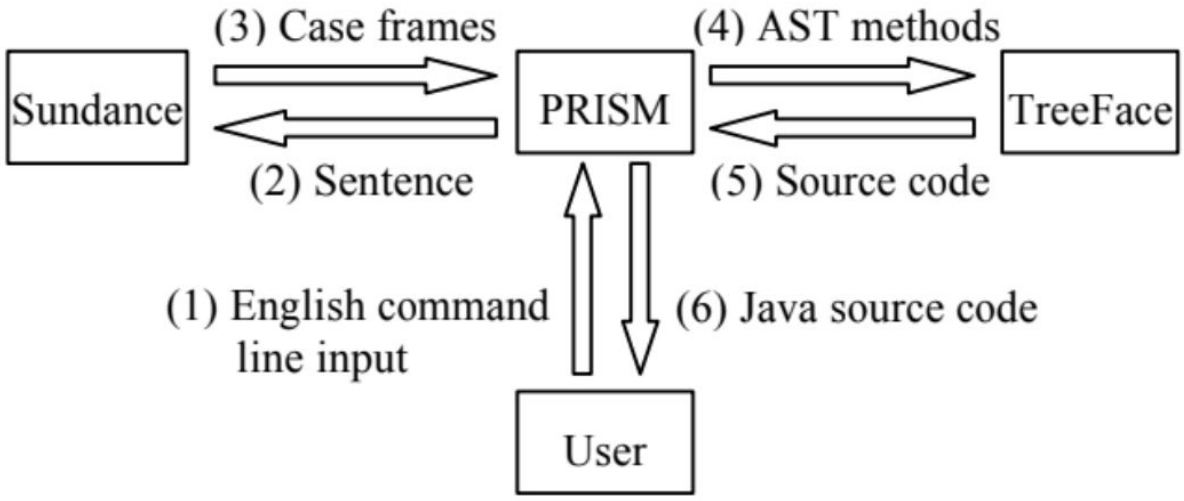


Fig. 3.1: Architecture of NaturalJava

3.1.1 Working

PRISM provides a command line interface to the user where he can key in his natural language commands. This input is passed on to Sundance which processes the input using natural language processing techniques and returns all applicable case frames to PRISM. A case frame represents a particular request. It is triggered by certain keywords and has slots which are filled from information in the input. PRISM selects the appropriate operation after looking at the set of case frames returned by Sundance and makes the edit/construct request to TreeFace. After each request TreeFace converts the AST into the source code and returns it to PRISM. PRISM displays this as output in the command line interface.

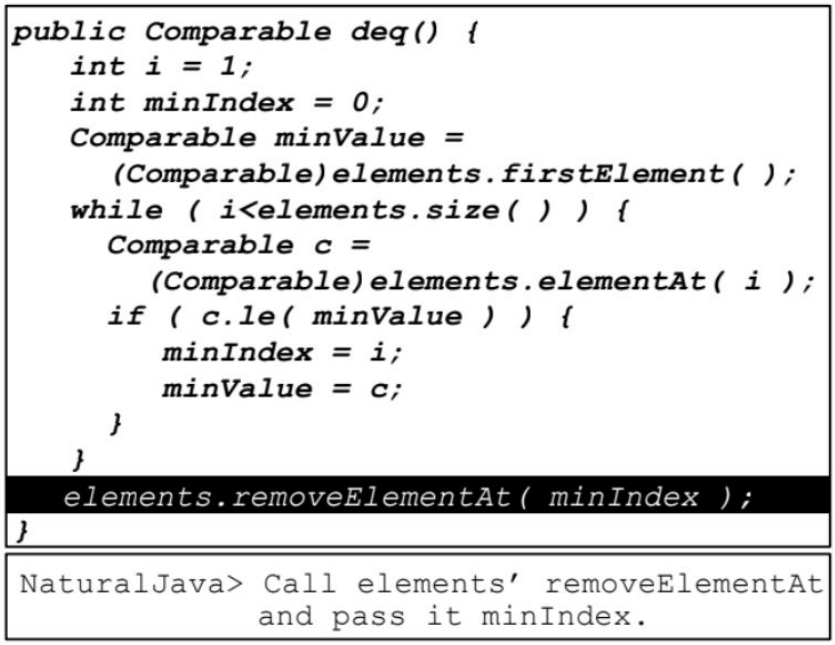


Fig. 3.2: NaturalJava's program display and user input windows

3.1.2 Limitations

- Speech input is not supported as speech recognition was not as developed at the time of its publication as it is today.
- Expression level editing is not supported.
- Few constructs like arrays and classes are not supported. However, these can be supported with the addition of the appropriate case frames.

3.2 Spoken Programs

3.2.1 Introduction

A survey [2] was conducted to understand how programmers vocalize code. This is better than making programmers adapt to new interfaces. Based on the results of the survey, a new dialect of Java called SpokenJava, easier to read out loud compared to traditional Java, was designed.

```
for(int i = 0; i < 10; i++) {
   x = Math.cos(x);
}
```

(a)

```
for int i equals zero
        i less than ten
        i plus plus
   x gets math dot cosine x
end for loop
```

(b)

Fig. 3.3: Part (a) shows Java code for a for-loop. (b) shows the same for loop using Spoken Java.

3.2.2 Results of the survey

- Spoken code is different from the code. eg. “array of i” → array[i]
- Partial words in the code are difficult to say. eg. println
- Spoken words can be converted to multiple valid variations of code.
- Pauses while speaking is used to disambiguate between different possibilities. eg. array[i++] and array[i]++.

3.2.3 Limitations

- Lacks flexibility as utterances have to be exactly as prescribed by SpokenJava.

3.3 VoiceCode

3.3.1 Introduction

VoiceCode [3] is an open source system which uses a natural language based command language for programming by voice. Natural language commands are translated into the syntax of the target programming language by the system. Same natural language commands can be used to program in different programming languages. VoiceCode also supports editing and navigation commands.

3.3.2 Features

- Code dictation and navigation can be mixed within the same command.
- Issues of punctuation and indentation are tackled by using code templates.
- Symbols can be dictated naturally and the system will find the closest names from a list of predefined symbols.
- Some commands can be dictated in multiple ways, giving some flexibility to the user.

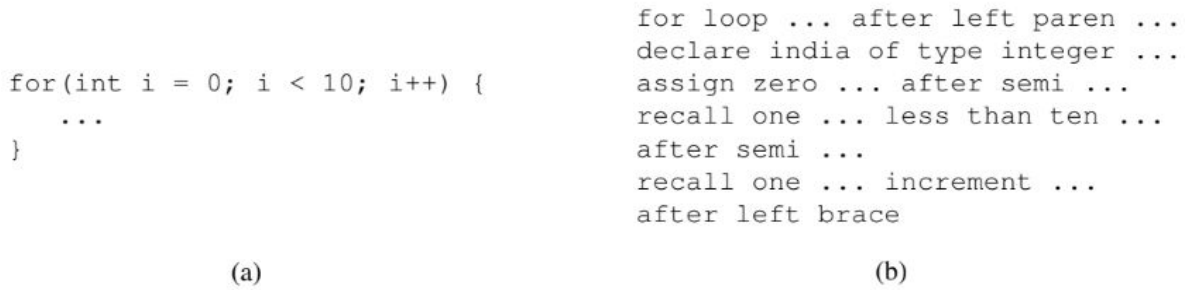


Fig. 3.4: An example of how to declare a for loop using VoiceCode.

3.3.3 Limitations

- Mixing navigation commands with code dictation is unintuitive at times.
- The software has not been maintained since its release and is almost unusable today.

CHAPTER - 4

PROJECT REQUIREMENTS SPECIFICATION

4.1 Scope

This project is among the first steps towards hands-free and voice coding, hence a simple version is implemented to test feasibility, upgrade in technology as compared to the few previous relevant works, ease and practicality of the idea. The assistant is made for coding in C language. It shall take only single action/operation as each input and process it, i.e., no conjunctions are allowed. The user may be asked to provide additional information when the given statement is insufficient or lacks certain objects or parameters to process it.

4.2 Product Perspective

The product will be available as an extension for the Visual Studio Code IDE. It will not only make it free, easily available and convenient but also easy to integrate for a huge section of Visual Studio Code users. Visual Studio Code had been chosen as it a fairly popular open source IDE with a huge community part of it, using it and contributing to it. It is also lightweight and highly flexible allowing programmers to have an editor which is light and can incorporate multiple functionalities into it customized to type or group of user.

4.2.1 User Characteristics

The assistant is intended for the use of programmers suffering from repetitive strain injury or with physical disabilities as the options they have are very restrictive and non-user friendly. It

could also be used where the use of a keyboard to program might be inconvenient, for example on mobile devices while travelling or even simply while teaching, allowing the user to move around freely and only bother about a mic. Developers who use multiple programming languages or are new to a programming language often have trouble remembering language syntax. A number of constructs, syntaxes, etc are repeated and reused, often resulting in silly and redundant mistakes, eg: missing brackets, semi-colons, etc. A natural language interface would remove the need to remember language-specific syntax and improve the user experience.

4.2.2 General Constraints, Assumptions and Dependencies

4.2.2.1 Constraints

1. Limited time restricts the scope of the project.
2. Unavailability of dataset constricts the system design choices, eg, no learning techniques applicable.

4.2.2.2 Assumptions

1. User is familiar with programming constructs in C and the terminologies used to describe them.
2. User is familiar with the English language.

4.2.2.3 Dependencies:

1. Speech recognition depends on Google cloud speech to text API. This requires internet connectivity.
2. The product is an extension for visual studio code IDE. Therefore the user must have visual studio code installed on his/her device and add this extension to his environment.

3. The product is also dependant on the user's accent and Google's speech engine output quality and accuracy.

4.2.3 Risks

The risks involved can be listed as:

1. User's English and accent, along with the output of speech engine.
2. Single statement input for simplification may cause inconvenience and unfriendliness.
3. Insufficient data restricting to a simple and not smarter version of the product.
4. Inability to complete all of the features due to time and complexity constraints.

CHAPTER - 5

SYSTEM REQUIREMENTS SPECIFICATION

Functional requirements of an application, drive the application architecture of the system, while the non-functional requirements drive the technical architecture of a system. A good set of both along with other requirements such as hardware requirements, software requirements, user interface external interface/communications, etc, determine the overall robustness, quality of design, simplicity, readability, writability, user-friendliness and ease of use and other such qualities.

The details of our system requirements specifications for our project are as mentioned below :

5.1 Functional Requirements

The Functional Requirements specify the operations and activities that a system must be able to perform. The Functional Requirements of our project are listed in this section.

5.1.1 User Requirements

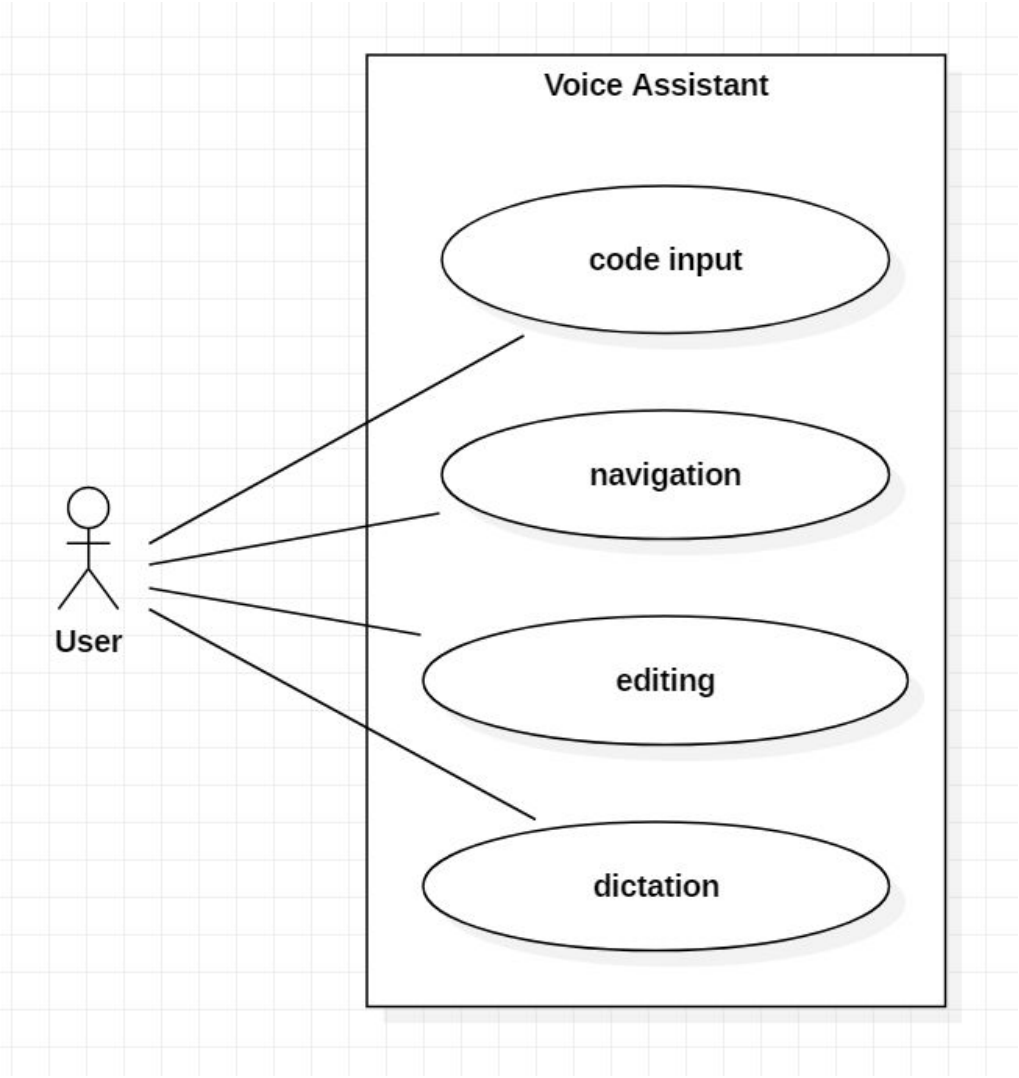


Fig. 5.1: Use Case Diagram

Use Case Item	Description
Code input	Input source code into IDE
Navigation	Move editing focus to a specified position in the program
Editing	Edit different aspects of the program

5.1.2 Input Data Description

The type of input being received by the assistant, being of speech type, introduces the problems and challenges of speech recognition faced today. However, technology has progressed enough for us to use it. It allows us to leverage all the advantages of voice inputs, eg. aiming for natural language imperative statements as inputs, or our intuitive navigational/directional commands.

The user can give various types of commands in a natural language manner. Currently, our system handles only the popular and basic constructs of C language, but the remaining can be easily added. This allows the user to say the constructs but in a more natural and intuitive way. Hence allowing him to code in C with natural language commands.

The user can also give commands to the assistant to perform various tasks, actions or operations for him/her, such as:

- Navigational - “goto line 4”, “go up”, “goto the end of line”, “go 2 places left”, etc
- Editing - “Undo”, “Redo”, “Delete sentence”, etc
- Additional - “Start Dictation”, “Exit block”, “Compile”, “Execute”, etc

5.1.3 System Workflow

In the backend, a number of tasks need to be performed to generate the output and/ or carry out the operation specified in the command provided by the user. The overall flow of our system and the works being done on the input may be described as follows :

1. The first step, receiving speech input, its processing and conversion to text is done by Google’s Cloud Speech to Text API. It will send a list of possible strings for the given voice input. Using confidence and scores of these strings, we select the best response and send it for parsing.

2. Next, the parser shall try to parse the given string in accordance with the provided language and its supported grammar and rules. On successful parse, next, a parse tree is created for the input, followed by its corresponding semantic generation.

3. The semantics is then taken by the executor and it tries to make sense and execute the command. The command type may be code generation, navigational, editing, etc. Hence, accordingly, appropriate actions are taken by the executor.

5.1.4 Output Data Description

There are several types of outputs being generated by the system. VS Code being our front end handles all outputs whether they are suggestions, errors, dictation output, etc. If it is a code insertion/ modification statement the editor is modified. Alternative suggestions for the code is provided in the suggestions pane. If the input is more of a command to the assistant, the executor determines the action and Visual Studio Code performs it(update cursor, etc). Dictation outputs are displayed in the dictation pane and may be discarded or appended into the editor. Errors encountered across the system will be collected and reported as voice feedback and also logged in the errors pane in our panel. The corresponding input which caused the error shall be displayed in the suggestions pane and other information about the error type, state of the input and how much has been translated shall also be displayed if available.

5.2 Non - Functional Requirements

Non-functional requirements are often called the "quality attributes" of a system. Some of the non-functional requirements of the assistant are as mentioned below :

5.2.1 Platform compatible

It is available as a Visual Studio Code extension, and Visual Studio Code is compatible across a range of platforms, i.e. Linux, Windows, Mac, allowing it to work with more than one hardware platform or operating system. Compatibility with various systems helps in developing a large community for the assistant (reaching the entire user audience).

5.2.2 Accessibility

A major advantage of the entire project is that it itself is a tool to improve accessibility to coding. A large number of programmers today, throughout the world suffer from repetitive strain injury, carpal tunnel syndrome, etc. as they are coding for very long periods of time on their computers. Also, there are so many people who are physically challenged and handicapped hence cannot type but have tremendous knowledge and skills in coding. Few software which exist allowing users to code vocally, expect the user to adapt to them and their completely new language, grammar, etc. However, this project tries to give the user as much leniency as possible and not only make a user-friendly assistant in terms of ease but also simple, easy and fast to make it attractive for all programmers too.

Deploying it as a Visual Studio Code extension allows us to easily access all Visual Studio Code existing users and spread our project easily. Availability as an extension allows users to use all other features of the IDE according to their personal preferences and simply adding the assistant as a small extension in their existing IDE.

5.2.3 Extensibility

Software systems are long-lived and will be modified for new features and added functionalities demanded by users, extensibility enables developers to expand or add to the software's capabilities and facilitates systematic reuse. Adding features and the entire set of C

language constructs, and carry-forwarding of customizations at the next major version upgrade is easily possible in our architecture.

Modular layered structure lets the entire codebase be highly readable and easily modified to add remaining C language constructs and features or even change the rules to accommodate rules for another language and use the modded assistant. The assistant can be extended later onto a mobile editor app too.

5.2.4 Performance

Our project is one of the first in this direction aims to get useable software and later enhanced and optimized. The current expectation for a response is at max 4s. The performance of the system is affected by the following factors :

- Language and speech clarity of user
- Internet
- Google Cloud Speech to Text API
- Ambiguities in the given input

As the popularity of the assistant increases, various additional constructs and features will need to be added. Ample amount of scope exists to improve the system and its performance. However only the major factors have been handled in this project as the focus of the project is not to develop a highly optimized and very high-performance software but to see can it be implemented, how much can be implemented, the cost in terms of money, time, effort, results, etc.

5.3 Hardware Requirements

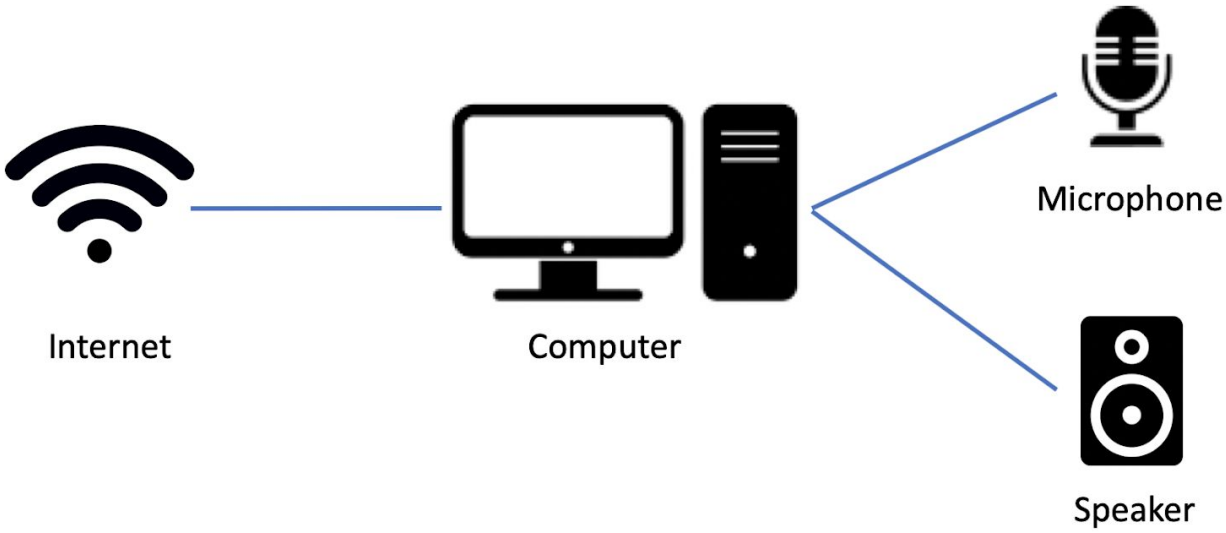


Fig. 5.2: Hardware Requirements

This project has minimalistic hardware requirements making it more user-friendly and simple. It also avoids unnecessary dependencies and delays. The hardware requirements are as follows :

- A Computer / Laptop
- Built-in / external microphone
- Internet
- Speakers

5.4 Software Requirements

The software requirements for this project are as follows: A simple OS which comfortable supports VS Code shall be sufficient from the software aspect.

- A simple OS (supports VS Code)
- Visual Studio Code
  - A cross-platform IDE
  - version 1.31.1 or above
- Google Cloud Speech Engine
- Python Interpreter
- Python Libraries
  - google cloud speech
  - pyaudio
  - word2number
- TS Library
  - child_process

5.5 User Interface

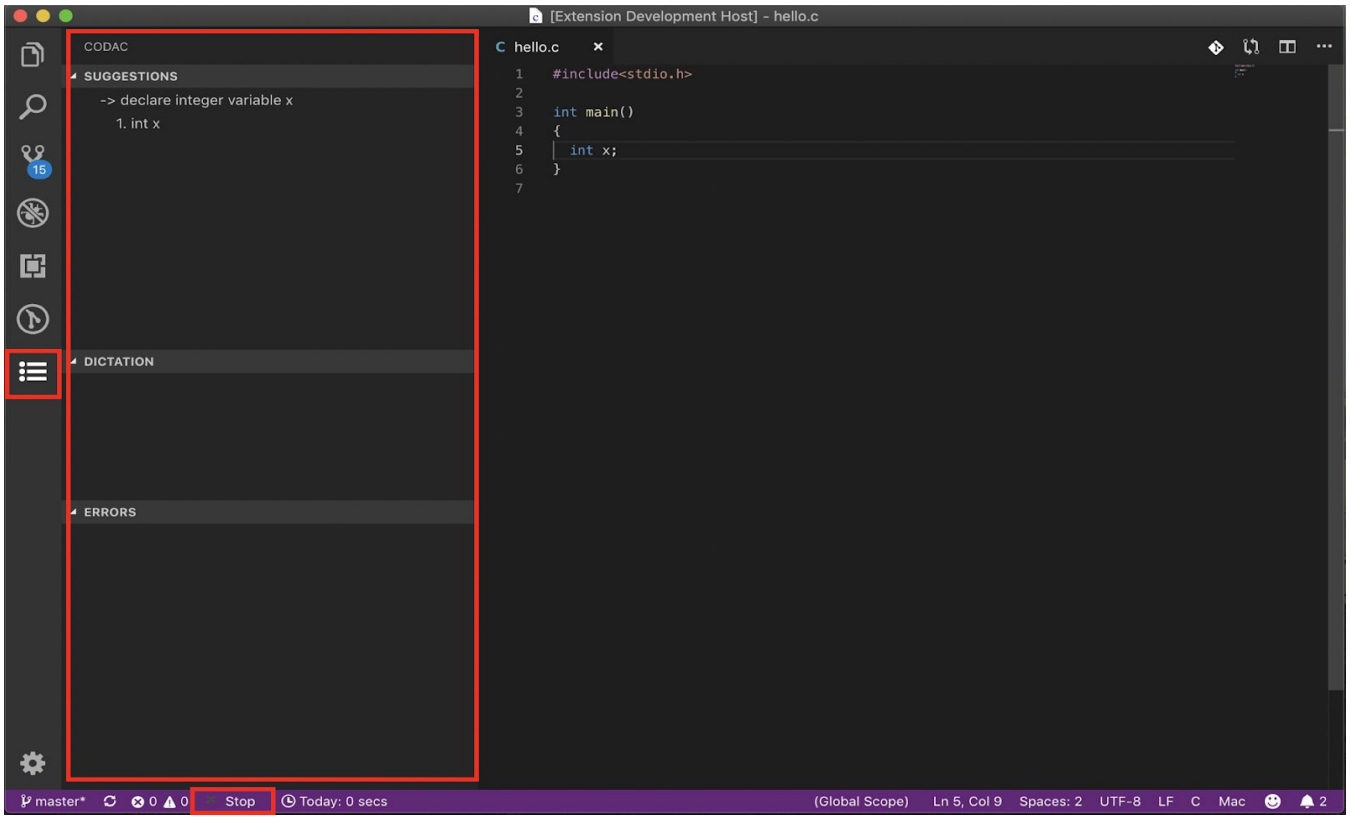


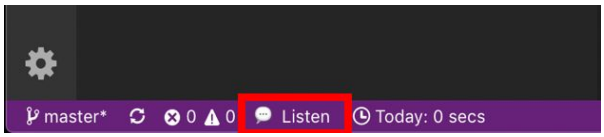
Fig. 5.3: The UI and its elements

The user interface adopted for this project is the Visual Studio Code IDE itself. This selection is apt as Visual Studio Code is highly flexible and coder-friendly. It gives complete control of the interface, provides popular actions as predefined commands and also there exists ample amount of documentation and codes online to easily use it. It is also a good choice as VS Code is one of the popular IDE's and gives the user a familiar environment instead of a new IDE or editor.

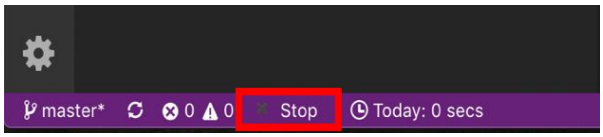
The primary components, concerned with our extension are as follows:

- The editor in which the codes are inserted and(or) modified. The editor is simple and shall be according to the user's personal setting in VS Code.

- The status bar at the bottom of the editor window will have a button “listen” which will start the voice assistant. At any point, it shows whether the assistant is listening or is off. When the assistant is active it will turn into “stop” and will stop the assistant on click. The same has been depicted in the figures below :



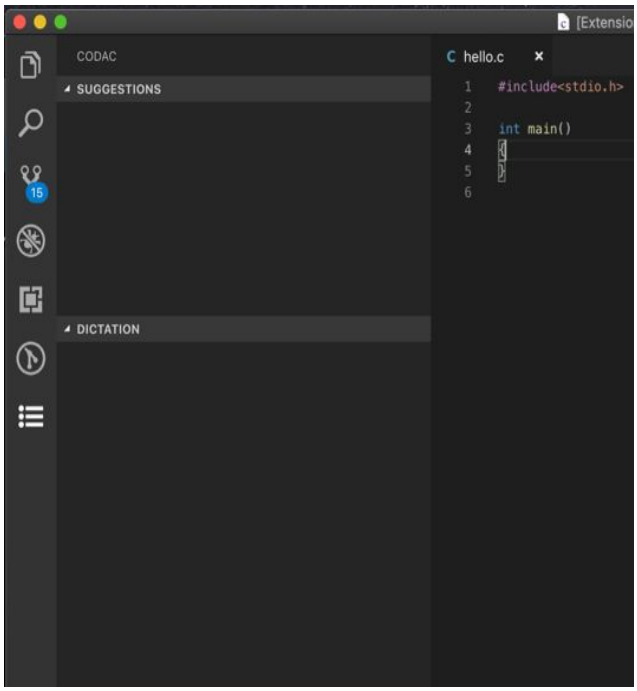
(a) Initial State



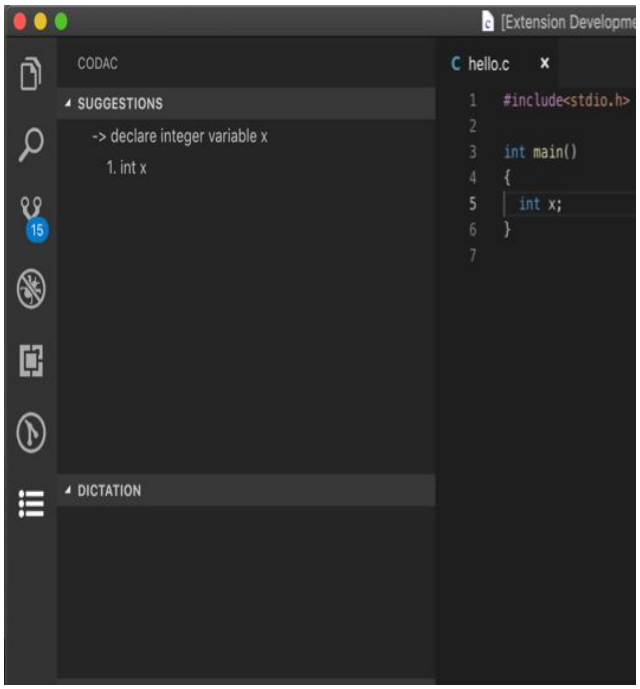
(b) On Click of Listen button

Fig. 5.4: States of the assistant’s status button

- The activity bar on the left will contain an icon for the extension as shown in the figure. On clicking it will expand from the left to reveal a view(panel) which will display additional information that the assistant provides. It has primarily three sections :
  - Suggestions Pane -The input and list of alternate suggestions are displayed here.



(a) Before and during processing



(b) After processing

Fig. 5.5: States of the suggestions pane

- Dictation Pane - It has a button which on click will start dictation and show the text there in the pane and later into the editor.

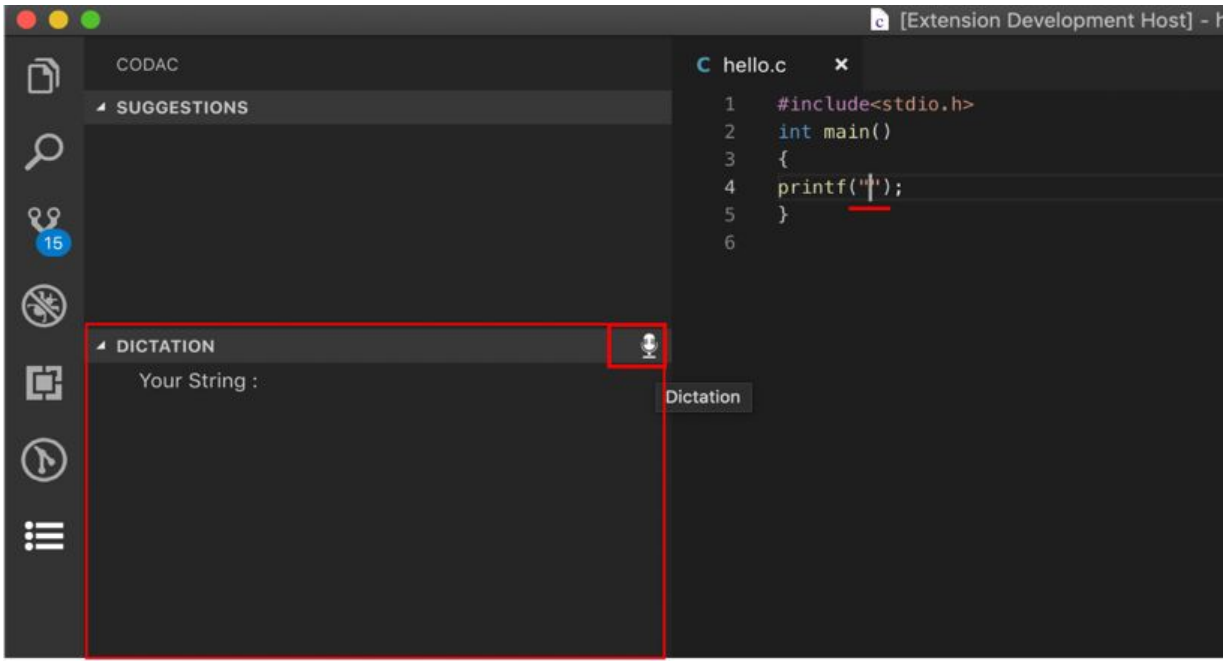


Fig. 5.6 : (a) On start on dictation(click on mic button)

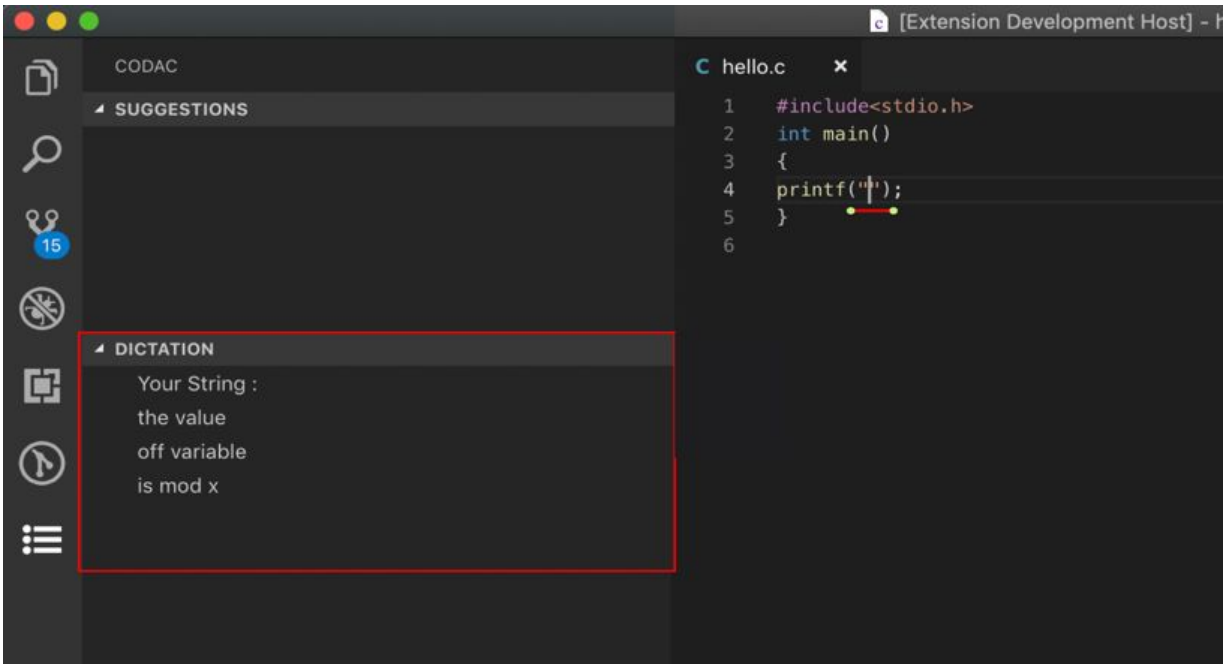
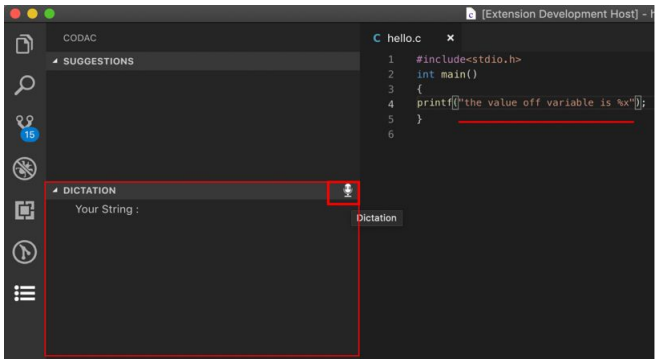
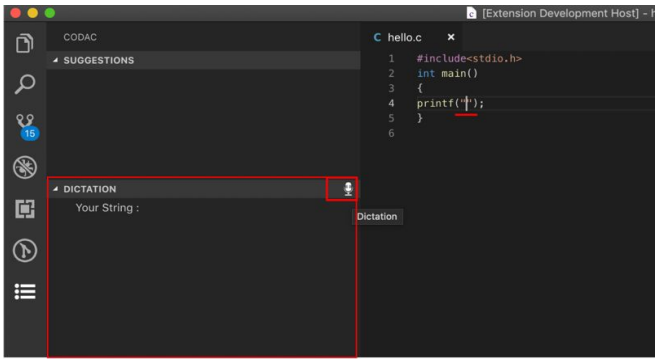


Fig. 5.6 : (b) After getting the required string



(i) Success



(ii) On exit

Fig. 5.6 : (b) On the end of dictation if success

Fig. 5.6: States of dictation module

- Errors Pane - It shows the list of errors detected.

- Notifications will be shown on the bottom right as popups which will disappear after a while.

5.6 Communication Interface

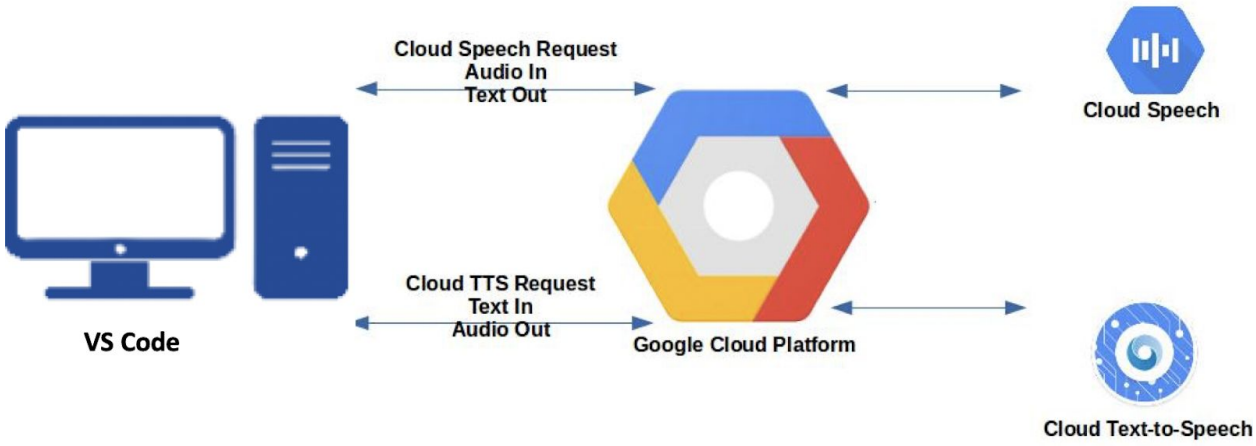


Fig. 5.7: Communication Interface

The app on the users' device shall primarily communicate with the Google Cloud Speech API as the rest will be provided to him as a bundle making it independent, secure and a lot faster. This shall reduce dependencies and make the system faster. Hence a good speed internet connection will make the entire user experience much richer and better.

5.7 Help

The help and support that shall be available to the end users shall be initially documentations and references/help. The simplicity hoped to have attained should require minimum learning for the user. The architecture and overall design were made with the goal to parse, our intuitive natural language thoughts. The user is expected to have prior knowledge of C language and its constructs. However, natural language inputs give the user leniency in terms of exact syntax.

Since it shall be available as an open source VS Code extension, it shall be shared to the entire community creating its own community, which shall not only grow to help and support but also promote and develop.

CHAPTER - 6

SYSTEM DESIGN

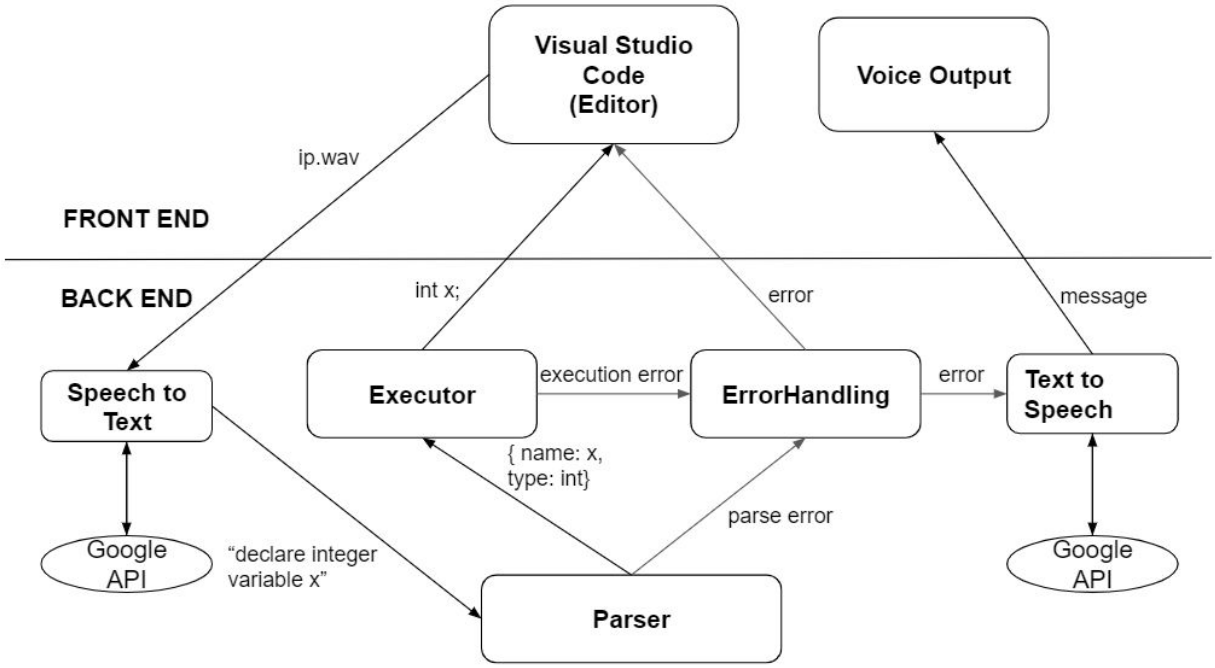


Fig. 6.1: System Design

6.1. Front End

The front end of a system is primarily concerned with the User Interface. Our system is comprised mainly of two modules. They are :

6.1.1 Visual Studio Code

The interface for the user is the visual studio code IDE which will have our software installed as an extension called ‘codac’. It shall control and manipulate the editor and its elements. The generated codes, suggestion, errors shall all go to Visual Studio Code and then to editor.

6.1.2 Voice Output

Apart from the visual feedback received in the IDE, the user will also receive voice feedback from their speakers. For the time being these are in the form of error messages and requests for additional information required to carry out requests.

The voice outputs and feedbacks that the assistant shall be taking is the first step in truly making it interactive and hopefully smart. Thus our project not only demonstrates the feasibility of the idea but also the direction and guidelines for starting off in this direction.

6.2 Back End

The backend of our system has been divided into the below-mentioned components. This promotes modularity, readability and writability in code and allows us to focus on one without affecting the others.

6.2.1 Sequence Diagram

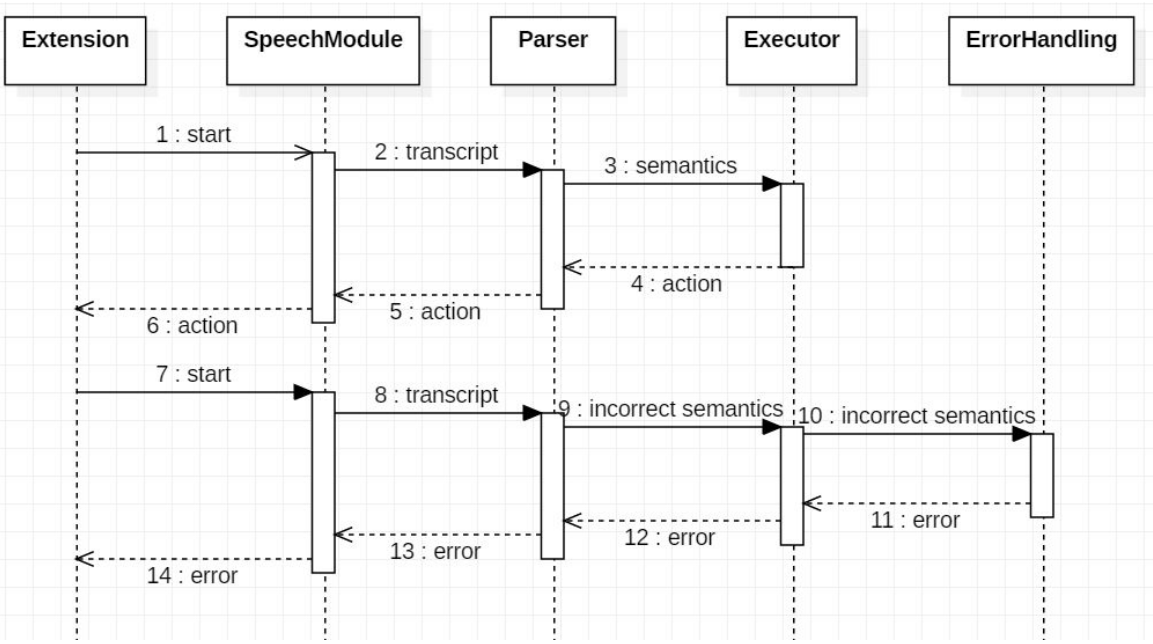


Fig. 6.2 : Sequence Diagram

6.2.2 Speech to Text

The command to be received from the user is through the microphone an audio/ speech input and we would like it to be a string. The Speech to Text module handles all such tasks. It makes the appropriate calls to the Google Cloud Speech-to-Text API. Receives the transcription(responses), selects the one with the highest confidence amongst them and passes it onto the next module, the parser.

6.2.3 Parser

The parser takes the transcription as input and generates all possible parses in accordance with the provided grammar. It also produces the semantics. All the parses that were generated are then scored according to their relevance, and the top few are sent to the executor to perform the

action requested by the user. Inability to parse the provided input according to the grammar will call functions within the ErrorHandling module.

6.2.4 Executor

The executor takes the most relevant semantics provided by the parser and determines the action to be performed. Various types of command may be passed such as code generation, navigational, editing, etc., therefore, appropriate actions are taken by the executor to check, verify and execute.

6.2.5 ErrorHandling

Various types of errors can be encountered in any system. In ours, they may occur in the speech module, parsing, execution, internet, etc. ErrorHandling module is responsible for handling different kinds of errors that arise anywhere in the system and make sure they are reported in the right place in the right format while allowing the system to function normally and smoothly.

6.2.6 Text to Speech

Apart from seeing the errors on the screen, it might aid some users to get voice feedback for the errors. This is achieved by the Text to Speech module that calls Google cloud Text-to-Speech API to convert the error message to speech.

CHAPTER - 7

DETAILED DESIGN

7.1 Master Class Diagram

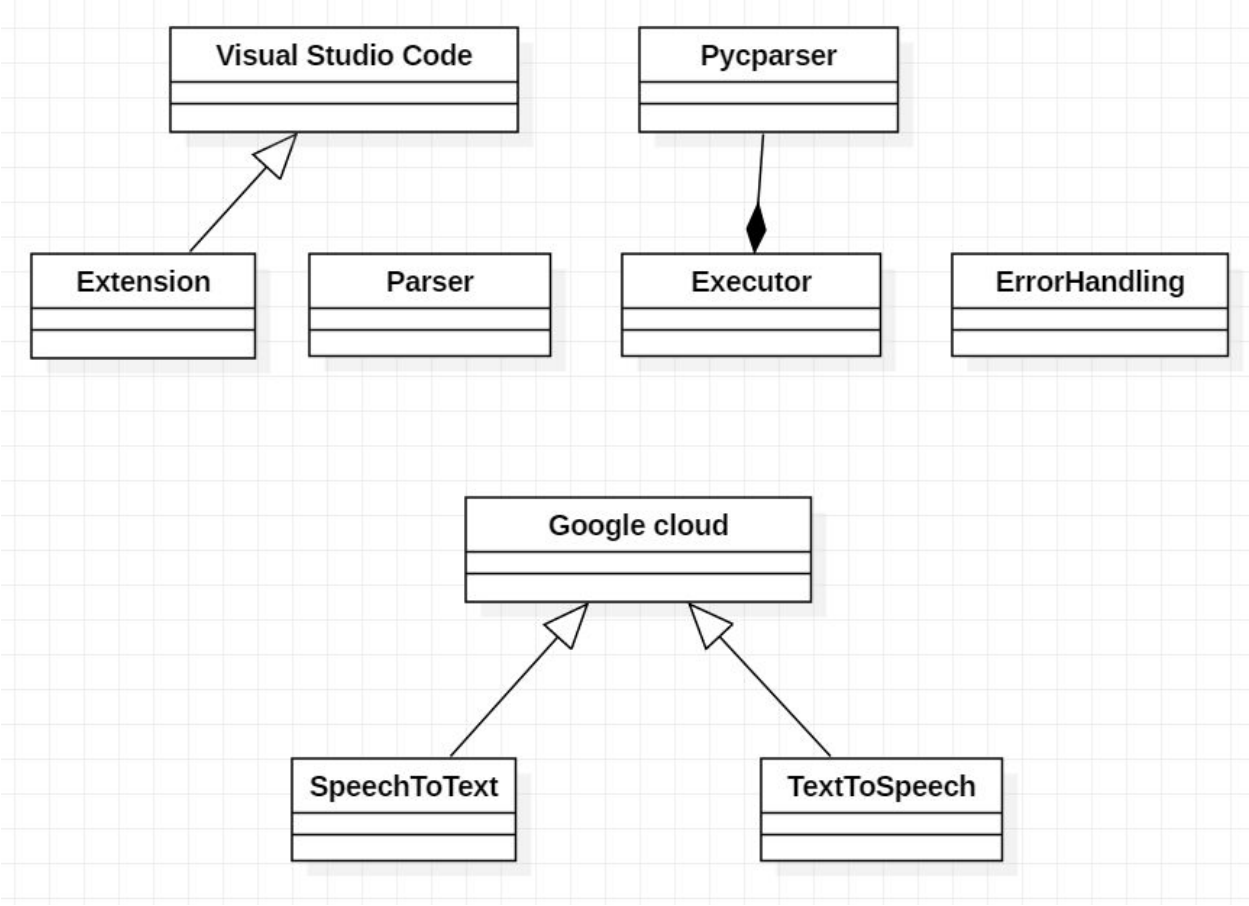


Fig. 7.1 : Master Class Diagram

The master class diagram of our system with its constituent modules has been depicted in the above figure.

7.2 Visual Studio Code Extension

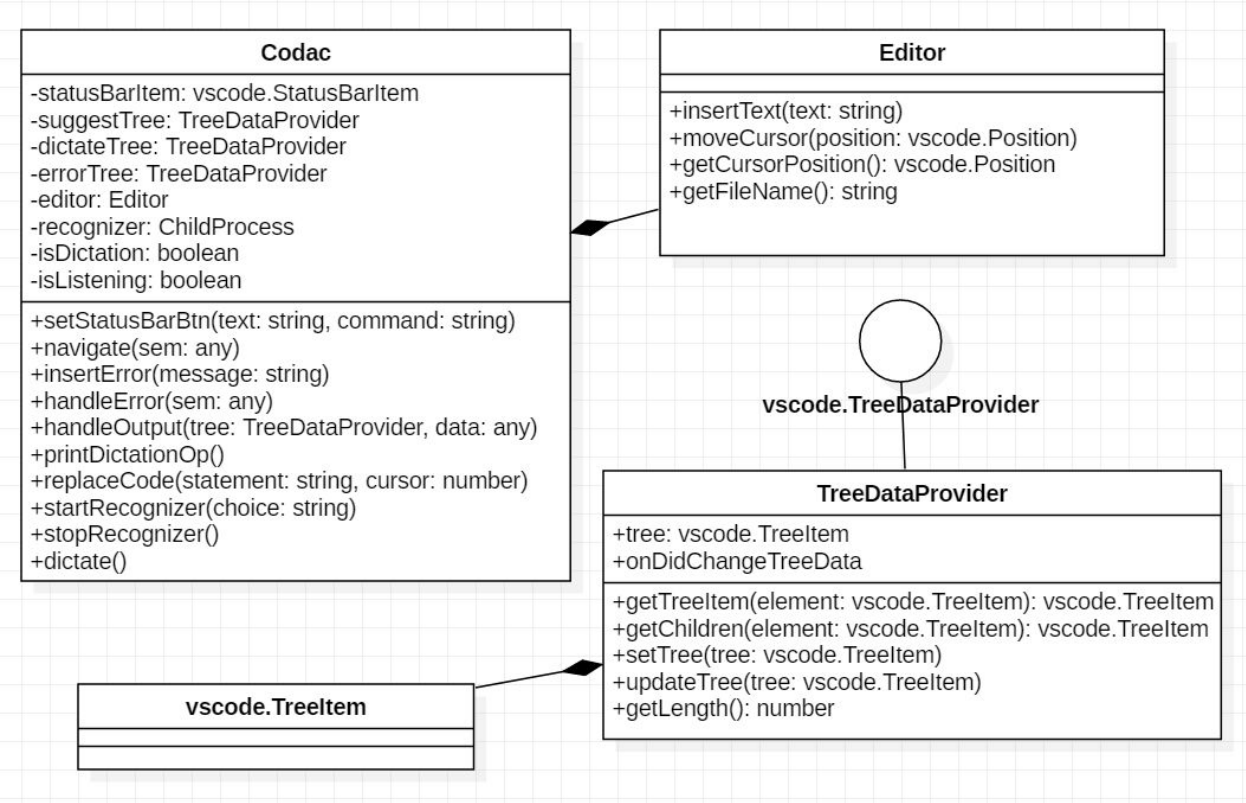


Fig. 7.2: VS Code Extension Module Class Diagram

7.2.1 Codac

This is the controlling class that has functions to start and stop the recognizer. It performs all the major function of the extension with help from the other classes.

7.2.2 Editor

The editor is interface class to the currently active editor in Visual Studio Code. It provides various operations that make changes to the editor.

7.2.2 TreeDataProvider

TreeDataProvider is an implementation of the vscode.TreeDataProvider interface provided by Visual Studio Code to supply data to the side panels.

7.3 Speech to Text

The Speech to Text module is a very simple script that makes an API call to Google cloud to get the transcription of the speech input provided.

7.4 Parser

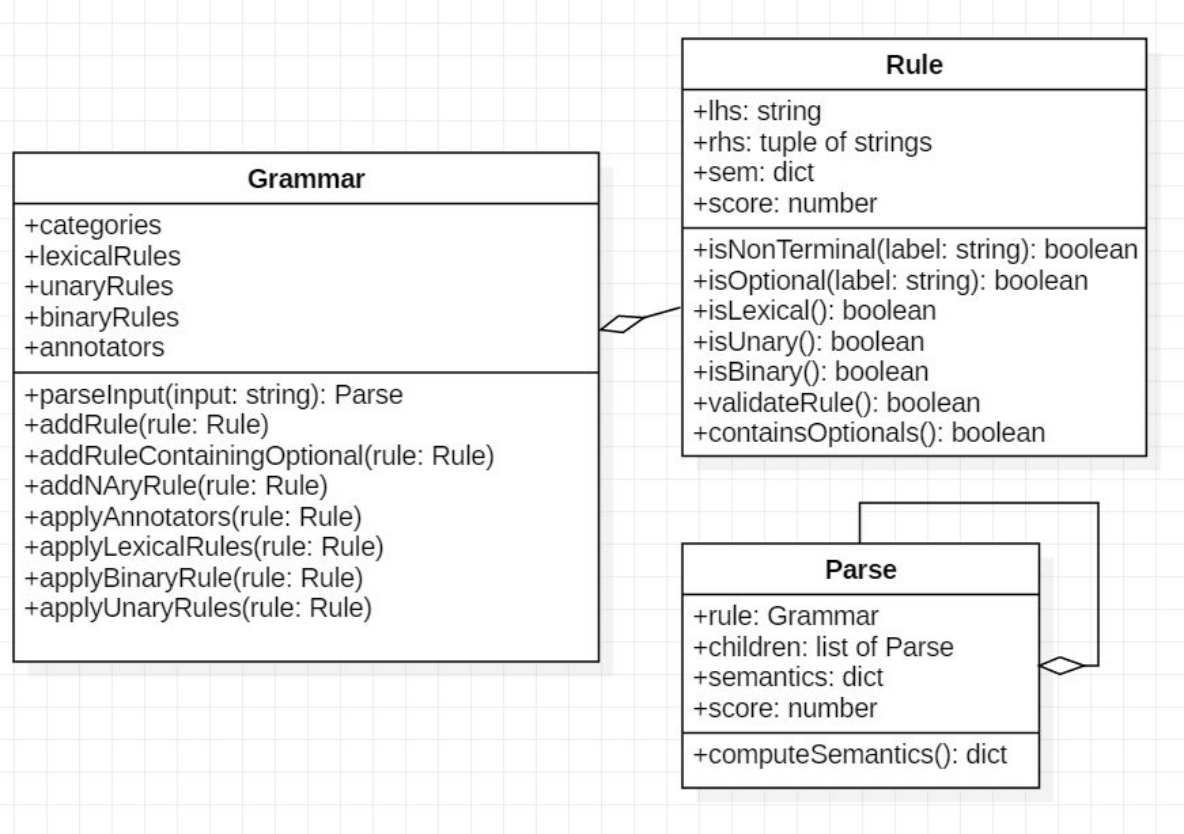


Fig. 7.3 : Parser Module Class Diagram

7.3.1 Grammar

This is the main class that has the parseInput method that is used to the parse the input provided by the extension. It holds different categories of rules and has operations to apply these rules on the input to produce parses.

7.3.2 Rules

Rules are the basic building blocks of the grammar. They have a left-hand side, right-hand side, a semantic and a score. There are different kind of rules, viz. lexical, binary and unary rules.

7.3.3 Parse

Parse is the type of parses that are generated by the parser. It has a tree structure which it achieves by having attributes of its own type. Each parse also gets a score based on rules that are used to generate it.

7.5 Executor

Executor uses the pycparser python package to convert the code to its AST representation, manufacture and insert nodes into it, then convert it back to the source code.

7.6 Error handling

ErrorHandling collects all the errors received from the different parts of the system and decides where and how each error has to be reported. The error is reported in the errors pane in and the input is displayed in the suggestions pane. Certain types of errors will have additional information like semantics, system message, etc when available to further improve the overall system and ease of use.

7.7 Text to Speech

The error handling module shall call the text to speech module to generate audio pieces of the error messages to be provided to the user.

CHAPTER - 8

IMPLEMENTATION AND PSEUDO-CODE

8.1 Visual Studio Code Extension

The Visual Studio Code extension is implemented in TypeScript which is later transpiled into JavaScript. Visual Studio Code commands are used to start and stop the recognition process. The side panels are populated by specifying data providers for each of the suggestions, dictation and error panels. Editor operations like inserting source code, navigating to a particular position etc. are achieved through inbuilt commands provided by Visual Studio Code. Some of the configurations and UI elements are listed in the package.json file.

8.2 Speech to Text

The Speech to Text module uses Google Cloud Speech Client library to make API calls. It sets up various configurations necessary to make the recognition request to Google cloud speech to text engine. Voice input from the user is streamed from the microphone to the server by means of a streaming recognition request which is terminated automatically by the recognizer when it detects that the user has stopped speaking. Some of the key parameters in the recognition configuration are :

- max_alternatives - determines the number of alternatives for the transcripts that are returned as responses.
- phrase_hints - an optional list of phrases that tend to occur more often in the current domain may be provided along with the request to improve recognition results.

8.3 Parser

The parser is implemented as a bottom-up chart parser that uses a provided grammar. A chart parser generates all possible parses of the input with the provided grammar. The grammar is represented as a set of rules, which have a left-hand side, right-hand side, semantic and a score. The grammar also employs annotators that tag tokens such as numbers and variable names. The grammar determines the types of input the assistant shall accept, i.e., its language, therefore a good set is crucial to developing reliable and useable software. On the generation of all possible parses, the total sum of scores of all the rules used to generate each parse becomes the score of that parse. Next, the parses are then sorted according to their scores conveying that the most probable semantic is at the top. The semantics of the top few parses are picked and provided to the executor to perform the user's request.

8.4 Executor

On receiving the semantics the executor determines whether the request is feasible or not. There are primarily three kinds of requests accepted by the assistant- code input, navigation and editing. For each of these requests, the executor works differently.

For requests that intend to input code, the executor makes use of the pycparser python package to perform certain functions. It manufactures the AST node by using a set of predefined templates and then proceeds to insert this node at the right position in the generated AST of the source code the user is trying to edit. Next, the AST is converted back to source code and sent back to the extension for insertion into the editor.

For editing and navigation requests the majority of the load is taken off the executor as editing and navigation are already supported by Visual Studio Code. A number of low-level commands have been predefined by Visual Studio Code and can be easily leveraged. Therefore

all executor has to do is relay back the semantics it receives from the parser to the extension. However, the executor still plays an important role in determining the action and types, finding the appropriate positions of constructs in the editor to successfully perform these actions and other such high-level tasks.

8.5 ErrorHandling

ErrorHandling looks at all the errors received from the different parts of the system and decides where and how each error has to be reported. Most of the errors are exchanged in JSON format The errors need to be properly propagated without crippling the system and crashing it. Each error is reported in the errors pane in the side panel and the input which had caused the error is displayed in the suggestions pane for a better experience and error tracking. Certain types of errors will have additional information like semantics, system message, etc when available to further improve the overall system and ease of use.

Taking accessibility into consideration, the error handling module in our project also interacts with a basic Text to Speech module, to give voice feedback for some of the errors or to request additional information from the user. A basic demo version has been implemented due to the time constraints but clearly conveying the feasibility and possible scope of the idea and the feature, given the appropriate time and resources to implement a better and more interactive model.

8.6 Text to Speech

The Text to Speech module is implemented using Google Text to Speech API (gtts). It accepts a string and additional information and returns a response. The response of the API is a

voice file in mp3 format. This temporary voice file is played back using operating system specific command line tools.

CHAPTER - 9

TESTING

Testing for this project was carried out for the different modules that make up the system. It was made sure that each module did its role. After integration of all the modules, two sample programs were written completely using voice to make sure the whole system functioned as expected.

9.1 Extension

The user interface of the extension has several elements. All the elements were tested in different scenarios to make sure they worked as expected, that is, they displayed the correct text, only the elements that were necessary for a particular scenario were displayed, the buttons triggered the right operations when clicked etc.

9.2 Speech to Text

Testing of the speech to text module is out of the scope of our testing as it relies mostly on an external API. However as it was critical to the functioning of our system, we tested the output of the Speech to Text module by giving input manually for a couple of examples of each category.

9.3 Parser

The parser has to be tested to make sure the grammar supports parsing all valid inputs and for valid inputs if multiple correct parses exist they have to be produced, sorted in the correct order. We made sure the parser gave the expected output for a list of test cases that we prepared. It was manually verified that multiple parses if they exist, are produced.

9.4 Executor

The output of the executor depends on the state of the program being edited. A sample program was generated that could cater to all the test cases and the output of the executor was tested against the expected output.

9.5 ErrorHandling

Testing of ErrorHandling happened alongside the testing of the Parser and the Executor. The test cases included the ones where there were missing fields, ones that could not be parsed etc. It was verified that the expected error was reported.

CHAPTER - 10

RESULTS AND DISCUSSIONS

On completion of this project, a number of technical and non-technical aspects are understood on a much better level than at the start. Some of the notable results and points worth pondering are as follows :

1. Although it is claimed that speech recognition has achieved a high level of accuracy today, we learnt that for domain-specific applications such as programming the recognition does not perform as well as expected. It performed poorly in recognizing identifier names and combinations of programming keywords. On further investigation, we learnt that this not due to the inability of the recognizer but due to the inability of the component of speech to text that actually maps recognized sounds to the text as it is trained to map the sounds to more commonly seen text. We were able to get better results by providing "phrase hints" to the recognizer. Phrase hints are phrases that are more likely to be seen in a particular domain, providing which increases the chance of them being transcribed. However, we believe even better results can be achieved by training a custom recognizer on a corpus that is close to the programming domain.

2. Using natural language commands to perform programming tasks is a highly feasible approach. However, odd constructs like format specifiers which are not usually vocalized have to be artificially mapped to natural language commands. Natural language coding is not only more intuitive and easy but also very alluring to all. Starting from the end user and his basic natural language commands and slowly increasing the set to accommodate more types of commands and languages may not be the most efficient but is definitely a reliable and implementable approach. Modern programming languages are moving towards more natural syntax and hence will not face this problem in the near future.

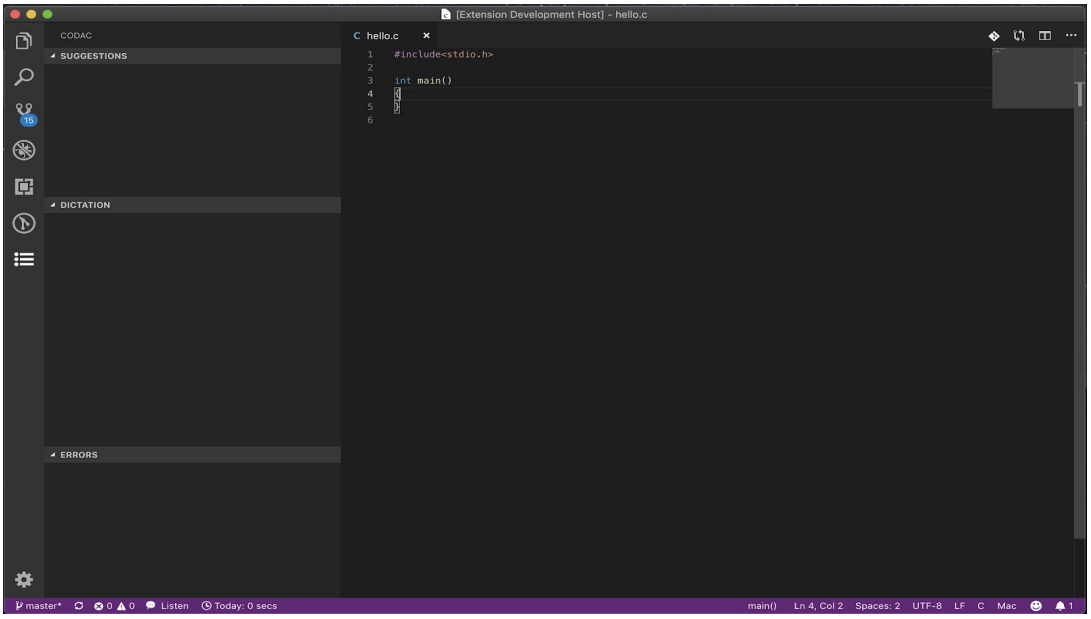
3. Developing extensions to already available IDEs is a great way to publish programming tools. This increases accessibility as users would get to experience the tool within an environment with which they are already familiar. However, extensions do have their drawbacks as they limit the flexibility to add user interface elements, affecting the intuitiveness of the interface.

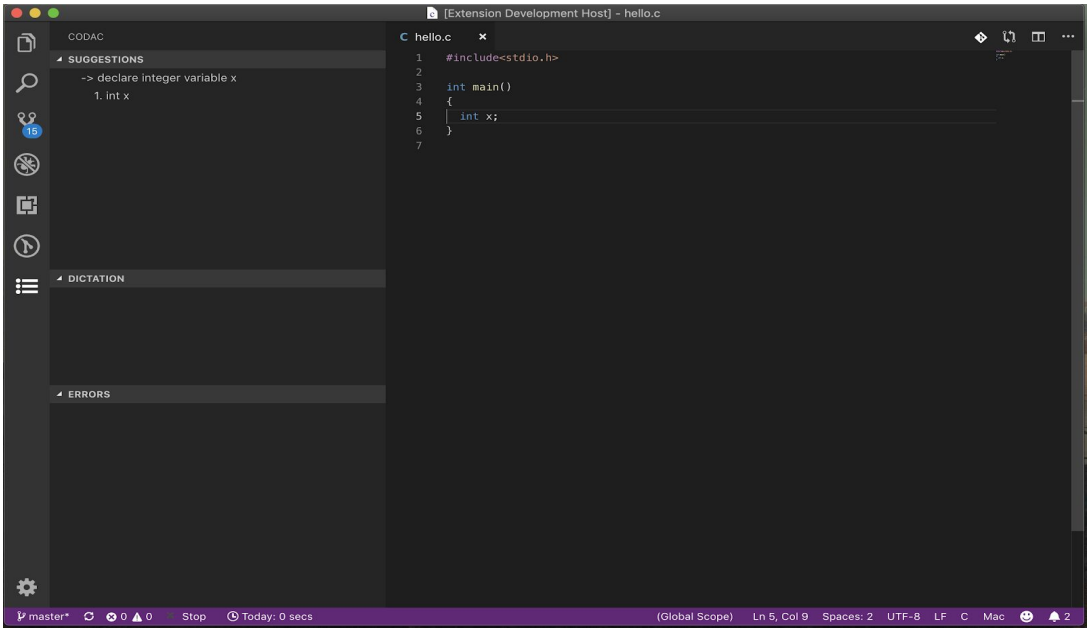
4. Natural language and speech are always associated with ambiguity. As our focus was not on speech, ambiguities in its direction have mostly been ignored. When it comes to the ambiguities in natural language, we resolve them based on context and keywords present in the input. However, if we are still unable to resolve the ambiguity, we select a default choice for the user and provide him/her with an interface that lets them instantly switch to any other alternative they want from a list of possible alternatives.

CHAPTER - 11

SNAPSHOTS

11.1 User Interface





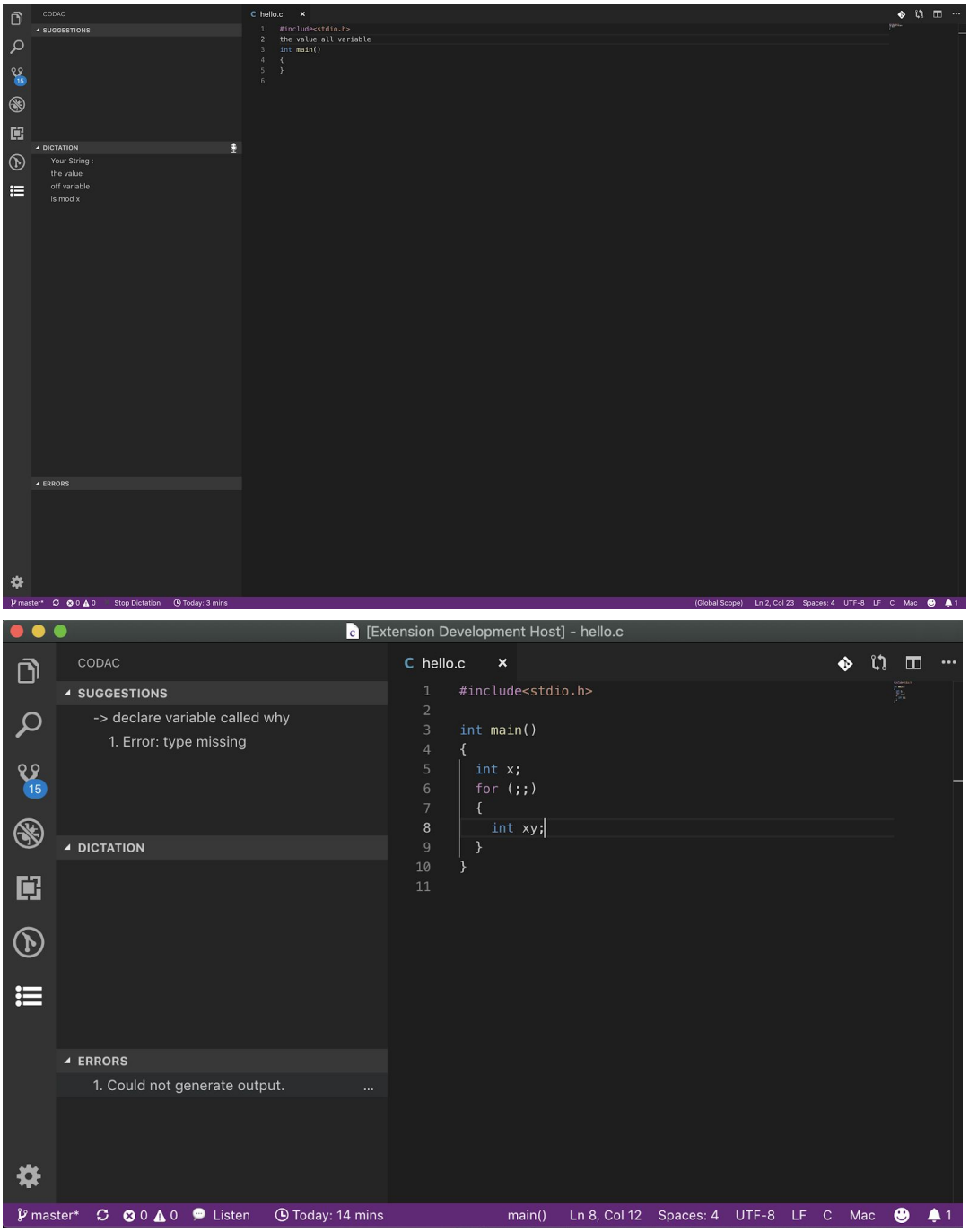


Fig. 11.1: User Interfaces Snapshots

11.2 Dataset

```
FORMAT:
*_ips = [
    [input: string, semantic: dict]
    .
    .
    .
    [...,...]
]
'''     VarunV-007, 7 days ago • added testing
arr_ips = [
    ['declare an array called x with type float of size 10',
     {'request': 'declare', 'construct': 'array',
      'name': 'x', 'type': 'float', 'size': (10, )}],
    ['declare an integer array x of size 10 by 20',
     {'request': 'declare', 'construct': 'array',
      'name': 'x', 'type': 'int', 'size': (10, 20)}],
    ['declare an array of type int called x of size 10',
     {'request': 'declare', 'construct': 'array',
      'name': 'x', 'type': 'int', 'size': (10, )}],
    ['declare an integer array of size 20 called x',
     {'request': 'declare', 'construct': 'array',
      'name': 'x', 'type': 'int', 'size': (20, )}],
    ['declare an array of size 20 of type int called x',
     {'request': 'declare', 'construct': 'array',
      'name': 'x', 'type': 'int', 'size': (20, )}],
    ['declare an array  called x of size 20 type int',
     {'request': 'declare', 'construct': 'array',
      'name': 'x', 'type': 'int', 'size': (20, )}],
    ['declare an integer type array called prog count',
     {'request': 'declare', 'construct': 'array',
      'name': 'prog_count', 'type': 'int'}],
]
```

```
exp_ips = [
    ['2 times 2',
     {'exp': ('*', ('value', 2), ('value', 2))}],
    ['2 times 2 plus 1',
     {'exp': ('*', ('value', 2), ('+', ('value', 2), ('value', 1)))}],
    ['2 by 2',
     {'exp': ('/', ('value', 2), ('value', 2))}],
    ['2 divided by 2',
     {'exp': ('/', ('value', 2), ('value', 2))}],
    ['x plus 10',
     {'exp': ('+', ('name', 'x'), ('value', 10))}],
    ['increment x',
     {'exp': ('++', ('name', 'x'))}],
    ['10 times 5 plus 1',
     {'exp': ('*', ('value', 10), ('+', ('value', 5), ('value', 1)))}],
    ['2 minus 1',
     {'exp': ('-', ('value', 2), ('value', 1))}],
    ['minus 1 minus 2',
     {'exp': ('-', ('-', ('value', 1), ('value', 2)))}],
    ['decrement y',
     {'exp': ('--', ('name', 'y'))}],
    ['x mod 10',
     {'exp': ('%', ('name', 'x'), ('value', 10))}],
    ['x modulo 10',
     {'exp': ('%', ('name', 'x'), ('value', 10))}],
    ['x by 10',
     {'exp': ('/', ('name', 'x'), ('value', 10))}],
    ['x cross y',
     {'exp': ('*', ('name', 'x'), ('name', 'y'))}],
]
```

Fig. 11.2: Dataset format and examples

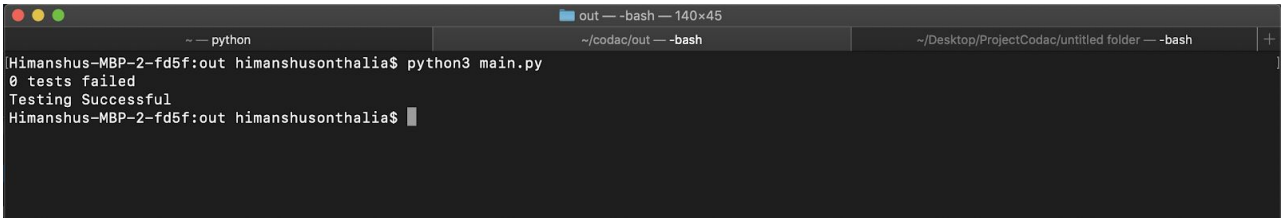
11.3 Google Cloud Speech to Text

```
Himanshus-MBP-2-fd5f:out himanshusonthalia$ python3 audio.py
{"status": "ready"}
results {
  alternatives {
    transcript: "declare an integer variable called X"
    confidence: 0.9558203816413879
  }
  alternatives {
    transcript: "declare an integer variable called EX"
    confidence: 0.9585577845573425
  }
  alternatives {
    transcript: "declare an integer variable call Dex"
    confidence: 0.9626093506813049
  }
  alternatives {
    transcript: "declare an integer variable called decks"
    confidence: 0.9647640585899353
  }
  alternatives {
    transcript: "declare an integer variable called Dex"
    confidence: 0.9647640585899353
  }
  is_final: true
}

Himanshus-MBP-2-fd5f:out himanshusonthalia$
```

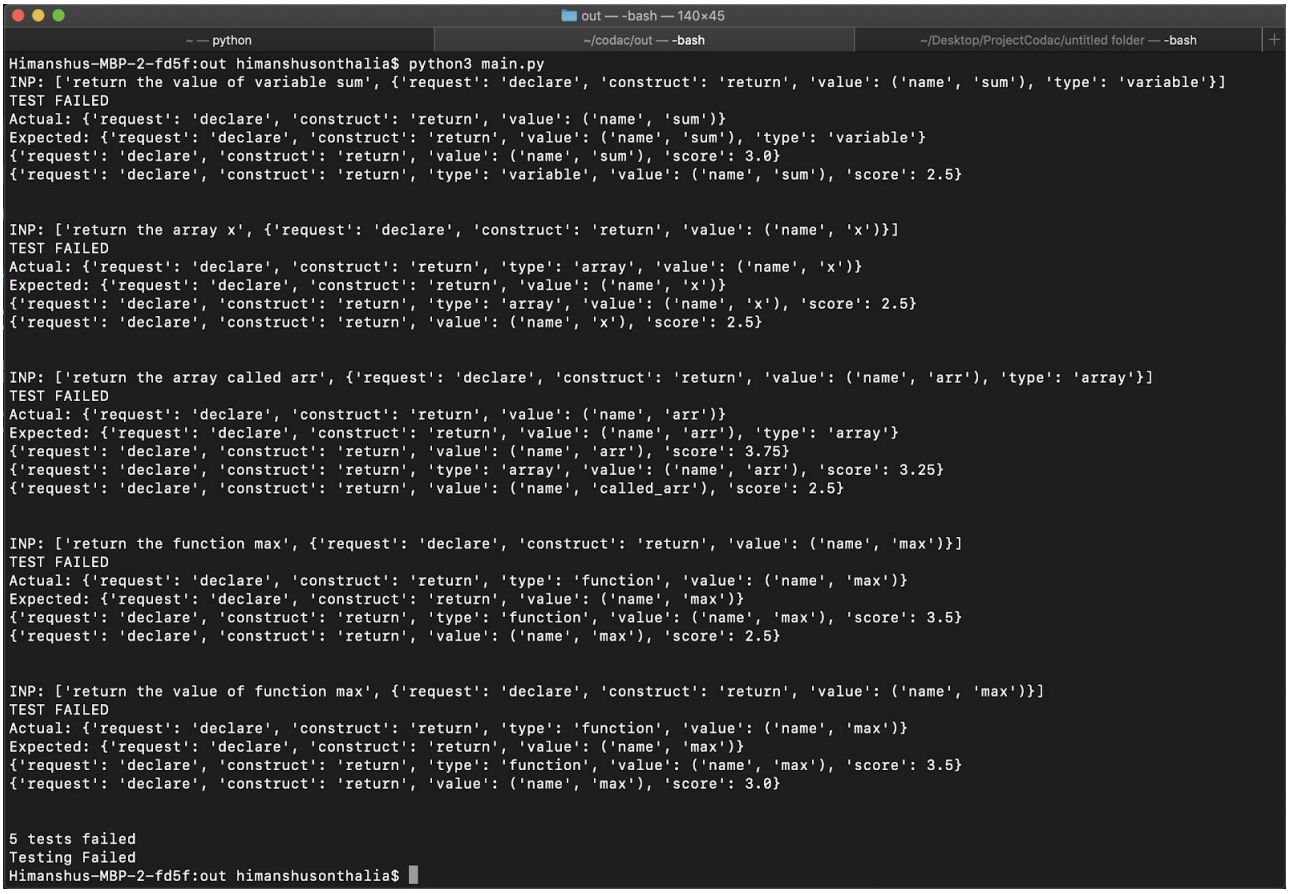
```
Himanshus-MBP-2-fd5f:out himanshusonthalia$ python3 audio.py
{"status": "ready"}
results {
  alternatives {
    transcript: "declare integer why"
    confidence: 0.9378986358642578
  }
  alternatives {
    transcript: "declare integer y"
    confidence: 0.9876291155815125
  }
  alternatives {
    transcript: "declare integer by"
    confidence: 0.9679945707321167
  }
  alternatives {
    transcript: "declare integer"
    confidence: 0.9039745926856995
  }
  alternatives {
    transcript: "declare integer Buy"
    confidence: 0.9679945707321167
  }
  is_final: true
}

Himanshus-MBP-2-fd5f:out himanshusonthalia$
```

Fig. 11.3: Google cloud speech to text response

11.4 Testing



(a)



(b)

Fig. 11.4: Testing : (a)Success and (b)Failed

CHAPTER - 12

CONCLUSIONS

Speech recognition and processing is yet one of the challenges today. However, it has progressed enough to be finally used well and leveraged enough to use its various features. Speech to text actually maps recognized sounds to text, as it is trained to map the sounds to more commonly seen text. Results improve by providing “phrase hints” to the recognizer.

Natural language coding is not only more intuitive and easy but also very alluring to all and more important this project clearly demonstrates its feasibility. Modern programming languages are moving towards more natural syntaxes and trying to free the user from language and syntactic restrictions.

Natural language and speech are always associated with ambiguity. As our focus was not on speech, ambiguities in its direction have mostly been ignored. However, with time, they shall definitely catch up to equivalently good models for the coding domain as for the natural language domain. When it comes to the ambiguities in natural language, they can be resolved based on context and keywords present in the input. Further, default values can be predefined, the user can be provided with an interface and a list of alternatives, that lets them instantly switch to other alternatives that they actually want.

In conclusion, we believe an even better result can be achieved by training a custom recognizer on a corpus that is closer to the programming domain. This will improve the quality of speech recognition and also pave the way to start works in the direction of learning based and NLP based techniques. They will hopefully further improve and enhance each module and the overall software too by finding patterns and various intricate details that might be easily missed. Numerous constructs and functionalities can be very easily added with minimum effort and a little time. In the end, we can say that coding through natural language with voice input is definitely a field worth exploring more and really promising in the near future.

CHAPTER - 13

FURTHER ENHANCEMENTS

The project has ample scope for improvements, enhancements and developments in all directions. They can be listed as below :

1. A number of the C language basic and popular constructs have been already added. Our modular design allows other programmers to easily add other C language constructs and complete the system and make it fully functional.
2. A number of minor enhancements and optimizations can be made to make the system highly efficient and optimized.
3. Additional functionalities such as debugging, file operation commands and other features which can be leveraged from voice technology can easily be integrated. This shall aim to improve the quality and support being provided by our assistant, therefore, broadly, targeting user experience enhancement features.
4. The system can be modified to accommodate/switch to another language altogether.
5. As the datasets collect and increase, using this system a smarter self-learning system can be developed.
6. As voice engine technologies improve, more support for different languages and other similar features can be added.

Voice engines may also be trained with the purpose of being integrated as a backup offline voice recognition engine.

REFERENCES/BIBLIOGRAPHY

[1] David Price, Ellen Rilofff, Joseph Zachary, and Brandon Harvey. 2000. NaturalJava: a natural language interface for programming in Java. In Proceedings of the 5th international conference on Intelligent user interfaces (IUI '00). ACM, New York, NY, USA, 207-211.

[2] A. Begel and S. L. Graham, "Spoken programs," *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, Dallas, TX, USA, 2005, pp. 99-106.

[3] Alain Désilets, David C. Fox, and Stuart Norton. 2006. VoiceCode: an innovative speech interface for programming-by-voice. In CHI '06 Extended Abstracts on Human Factors in Computing Systems (CHI EA '06). ACM, New York, NY, USA, 239-242.

REFERENCES/BIBLIOGRAPHY

1. https://medium.com/bambuu/state-of-voice-coding-2017-3d2ff41c5015
2. https://github.com/mozilla/DeepSpeech
3. https://cmusphinx.github.io/
4. https://www.typescriptlang.org/docs/home.html
5. https://docs.python.org/3/
6. https://code.visualstudio.com/api
7. https://github.com/Microsoft/vscode-extension-samples
8. https://nodejs.org/api/child_process.html
9. https://cloud.google.com/speech-to-text/docs/reference/rpc/google.cloud.speech.v1
10. https://github.com/wcmac/sippycup
11. https://github.com/eliben/pycparser

APPENDIX A

DEFINITIONS, ACRONYMS AND ABBREVIATIONS

The keywords given below are used in the document unambiguously. The meaning of these keywords remain as defined here unless indicated otherwise:

- AST - Abstract Syntax Tree is a tree representation of the source code.
- IDE - Integrated Development Environment is a software that provides an environment for software development which includes a source code editor.
- API - Application Program Interface is a set of functions that enable the program to access a certain service.
- OS - Operating System
- Visual Studio Code - An open source code editor available across Windows, macOS and Linux.
- Extension - In the context of Visual Studio Code, it is a way to extend the functionality of the editor to support other development activities.

Semantic - A meaning representation for a statement that would remain the same for another statement which is syntactically different but has the same meaning.

APPENDIX B

USER MANUAL

The below manual is a basic guide containing some of the types of inputs the user can give to perform different operations.

B.1 Code Input

B.1.1 Arrays

B.1.1.1 Declaration

1. declare an array called x with type float of size 10
2. declare an integer array of size 10 by 20

B.1.1.2 Indexing

1. value of x at index i
2. a of i

B.1.2 Assignments

B.1.2.1 Variables

1. set x equals 20
2. set the value of variable x to variable y

B.1.2.2 Pointers

1. set pointer x to pointer y
2. initialize pointer x to address of x

B.1.2.2 Array

1. set value of a at index j equal to value of a at index j plus 1
2. set a of i to 10

B.1.3 Conditions

1. a is greater than 10
2. x less than y and x greater than z

B.1.4 Expressions

1. x plus 1 times 2
2. x divided by 2

B.1.5 Functions

B.1.5.1 Declaration

1. declare a function called main having return type int
2. declare a function called fun

B.1.5.2 Parameters

1. add parameter x of type double
2. add parameter integer x

B.1.5.3 Calls

1. invoke the function ceil with parameter x

2. call function max and pass it arguments 2 and 3

B.1.5.4 Special Calls

1. call printf with string hello world
2. call scanf with parameters mod d and address of x

B.1.6 If

B.1.6.1 Declaration

1. declare if statement
2. declare if else statement

B.1.6.2 Adding else and else if

1. add else block
2. add else if

B.1.6.3 Adding conditions

1. add condition x equal 10
2. add condition x smaller than 10

B.1.7 Loops

B.1.7.1 Declaration

1. create a for loop
2. define a while loop

B.1.7.2 Adding initialization (only for-loop)

1. add loop variable i of type int

2. add integer type variable i set to 0

B.1.7.3 Adding condition

1. add condition i less than 10
2. add loop condition a is equal to 10

B.1.7.4 Adding update statement

1. add an update plus plus a
2. add loop step y plus plus

B.1.8 Package

1. include package stdio
2. include package called MyLibrary

B.1.9 Pointers

1. declare integer pointer ptr
2. declare a variable pointing to integer

B.1.10 Return

1. return 0
2. return a plus b

B.1.11 Variables

1. declare integer variable x set to 0
2. declare a variable called x of type integer

B.2 Navigation

B.2.1 Inline

1. go four places front
2. go three positions back
3. goto the start of line
4. go front one place

B.2.2 Across Line

1. goto line number 10
2. go to the next line
3. go 8 lines up
4. go 1 line down

B.3 Editing

1. Undo
2. Redo
3. Select current line
4. Delete previous line