Speak Pseudocode2c : A framework to convert customized pseudocode to c code

Prof. Nitin V. Pujari
dept. Computer Science and Engineering
PES University
Bengaluru, India
nitin.pujari@pes.edu

Shaashwat Jain
dept. Computer Science and Engineering
PES University
Bengaluru, India
shaashjain213@gmail.com

Raghav Aggarwal
dept. Computer Science and Engineering
PES University
Bengaluru, India
raghavaggarwal03.ra@gmail.com

Srishti Sachan
dept. Computer Science and Engineering
PES University
Bengaluru, India
srishtisachan31@gmail.com

Rajdeep Sengupta
dept. Computer Science and Engineering
PES University
Bengaluru, India
senguptarajdeep21@gmail.com

***Abstract*—In today's fast paced world, the education industry also has been developing at a rapid pace. This has introduced the world of coding to students at a young age. The source code of certain languages can be tough to understand and might make students drop interest in the subject altogether. The solution proposed in this paper addresses the simplicity of using natural language, spoken by the user and converting that to source c code using a stub based approach. The accuracy of converting speech-to-text using Google Cloud is 79% and for converting has been 100% as the approach is static and not Machine Learning based.**

***Index Terms*—Stub-based, XML(Extensible Markup Language), ML(Machine Learning), PLY(Python Lex and Yacc) Compiler, GCP(Google Cloud Platform) Speech-to-Text**

I. Introduction

Today, coding is being taught from primary levels at school. According to the new education policy, programming has been introduced as a compulsory course for students from junior classes.

Currently, there is very little support for students, especially beginners who are new to programming. They need assistance to understand the programming concepts better. With the surge in the number of programmers and advancements in technology, the increasing competition among students can be very intimidating for a child. Also, the c code given in textbooks is empirically observed to be not very easy for students to comprehend at the school level when compared to natural language pseudocode.

Pseudocodes are comparatively easy for students. They help in understanding the algorithm and flow of a program. Therefore, our goal is to use a voice-based digital assistant to automate the process of writing and editing code by providing our voice input.

Our aim for this project is to make these children familiar with the coding environment so that they do not shy away from programming and they have an all-time accessible tool to help them in their programming journey.

This tool will take a user's voice input as pseudocode and convert it into a c program, given that the user gives the correct pseudocode as voice input. The approach for converting spoken pseudocode to source c code is divided into 3 parts:

- Converting Speech to Text: This part of the framework takes voice input from the user and gives it to google cloud which provides us with generated text.
- Pre-processing text using NLP and digit conversions: This part takes care of proper formatting of the generated text. It also takes care of conversion of a few phrases to proper symbols eg - "less than" converts to " < ". NLP provides flexibility to the framework. Multiple phrases map to the same output. For Example:-
  "start the program" or "begin the program" will always map to -

```
#include<stdio.h>
int main() {
```

  Digit conversion ensures that when a user says a digit it always converts to its numerical form. For example - "one" to 1.
- Mapping formatted pseudocode to source c code: Mapping is a major part of our framework. It maps the

generated text to c language. The program generated is then preprocessed and is displayed in a formatted and clean code.

The **major contributions of this research work** are using stubs to map pseudocode to source c code and using intelligent classes and design approaches for tracking variables. Also, there is use of Natural Language Processing techniques for interpreting user conjunctions and preprocessing the language to give the subsequent output.

II. LITERATURE SURVEY

The task of converting speech to text requires extensive research into current language models in the market. One such study done by P. Sirikongtham and W. Paireekreng[1], uses speech recognition using dynamic multi-pipeline API. The study proposes a method to address the problem of choosing between different speech models. It combines 2 stages using Microsoft API and Google API model voice. The results suggest an increase of 21% from Microsoft's engine and 4% from Google's current engine and a significant decrease of 5-8% in the Word Error Rate(WER) of the combinational model.

Going further, the task at hand for converting pseudocode to source code posed many implemented models. The discussion further summarizes 4 approaches which have been used by different authors.

T. Dirgahayu, S. N. Huda, Z. Zukhri and C. I. Ratnasari, in their paper[2], proposes a conceptual metamodel approach which tries to do the opposite of what we are trying to do. The problem which the authors are trying to solve is to generate pseudocode from given source code automatically and to achieve this goal, they have built a neural network with the help of deep learning using Long Short-Term Memory architecture. They obtained 18,800 pairs of code along with pseudocode for their dataset. To measure the quality of generated pseudocode they used a scale called Bilingual Evaluation Understudy (BLEU). The obtained BLEU score is 54.78% which is better than other approaches.

Imam, Ayad Alnsour, Ayman, introduce a technique called Neural Named Entity Recognition from Subwords Units [1]. The model learns a low-dimensional representation from each subword unit (character-, phoneme-, and byte-level) for each word in an utterance, which is then concatenated and fed into a bidirectional LSTM-CRF model. We take in portrayals from every one of the three subword units for each word in an expression to interpret what they represent in converting pseudocode to source code. This gives substantial results and may be used as a part of the approach implemented in the paper.

The Use of Natural Language Processing Approach for Converting Pseudo Code to C Code [2], gives us an insight about a few techniques to enhance and make possible what the author calls the process to computerize the structure of a programming language code from pseudocode, which is seen here as an interpretation interaction for a characteristic language text, as pseudocode is an arranged book in normal English language. The first tool it talks about is an Automatic Code Generator, abbreviated as ACG, which uses programs to generate source code which humans would otherwise have to write. The paper also talks about the use of Natural Language Processing (NLP) techniques to format the input in various ways to take multiple inputs/slangs but generate the same source code which the user wants. The author proposes the "CodeComposer System", which is a Machine Translation tool that uses Semantic Role Labelling (SRL) and NLP techniques to identify and convert pseudocode to mapped C# code. An assessment of the exactness of CodeComposer utilizing a binomial method shows that it has an accuracy of 88%, a review of 91%, and a F-proportion of 89%.

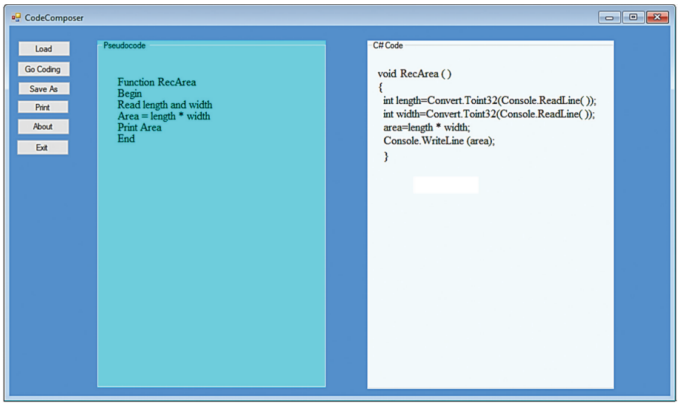

Fig. 1: The code composer system

The final paper reviewed [3] talks about the approach of trying to convert and edit pseudo code to source code in multiple languages by using XML (eXtensible Markup Language) and the features of DOM(document object model) by encapsulating features of each line of code in some tags example: $< var >, < var - type >, < var - name >, < var - value >$ etc. This model has the benefit of a systematic approach and the use of this converted document to be written in any language the user wishes and not just c code. The author mentions the use of $< cyc >, < while >, < for >, < do >$ and various tags for loops, and for blocks where even nesting is important, it introduces the ¡nesting¿ tag which can encapsulate various statements.

After reviewing all talked about papers, the evaluation resulted in a more static approach which uses function stubs to convert pseudocode to source c code. It uses a natural language processing middleware layer and further converts set natural language statically giving no error in conversion, but has a certain drawback of unsupported keywords, which is made flexible by using the NLP layer.

III. METHODOLOGY

A. Converting Voice Input to Text

1) Importing the libraries: We first import the google cloud speech to a text library which is used to send the audio data in the form of chunks. The authentication of the user is done automatically when the library is imported as it looks for a

JSON file which is generated for a service account and the path to that file needs to be in the environment variable.

2) Configuring the parameters: Since we are sending data in the form of chunks, therefore we need to set the frequency and language of the chunks. en-IN is used to recognize English-India which improves the accuracy of the output. The audio file is encoded with LINEAR16 (16-bit linear pulse-code modulation (PCM) encoding) codec. An audio encoding refers to the manner in which audio data is stored and transmitted. The output needs to be stored somewhere in order to do preprocessing on it. To do that a filename transcript.txt is used and the output is flushed from terminal to that file for further preprocessing.

3) Listening to the Microphone: As there is no limit on the duration for the user to speak, therefore the microphone needs to keep on listening. In order to do that we keep the mic open and when it receives the word exit or quit in the content stream, it stops the program automatically. MicrophoneStream is a class which contains modules/functions that acts as a generator for audio chunks by opening a recording stream. The API in this program supports only 1-channel (mono) audio. To fill the buffer object, the audio stream runs asynchronously. This is needed to prevent the input device's buffer from overflowing as the calling thread makes network requests

4) Printing the data received from server: To print the data which is received from the server we implemented the function called lis ten_print_loop which takes the responses as a function argument. The function Iterates through server responses and prints them. The responses passed is a generator that will block until a response is provided by the server. Each response may contain multiple results, and each result may contain multiple alternatives. We print only the transcription for the top alter native of the top result.

B. Converting numeric words to digit

We wrote our custom defined function to convert the numeric words to digit. For example: 1 one:1, two:2, five thousand six hundred seventy:5670 By doing this it will increase the readability of the pseudocode and it will help us in the later stage of the implementation where we have to map our custom pseudocode to the source code in c language. We have taken this a step further and also converted the basic mathematical symbols in words to actual symbols. For example: $Plus : +, Minus : -, Modulo :$

For this task we could have used the python libraries like *wordtodigits* or *words2number* but there was a problem with this. When speaking a word which can not be converted to a digit (normal english words) then it immediately stops listening to the server and returns an error which we don't want. Further these libraries do not map the mathematical symbols from words to actual symbols and in order to do that along with these libraries, it was increasing the latency which is again a downside. Therefore, using a custom defined function we can fill two needs with one deed.

C. Natural Language Processing

D. Mapping Pseudocode to Source code

1) Variable Tracker Module: It keeps track of variable's type and scope throughout the program for easy access to other modules and exceptions like Variable Already Declared and Variable Not Declared scenarios. It is a Singleton Class.

2) Input/Output Module: The module receives distinct types of inputs from the user i.e. declare, initialize, input (scanf) and output (printf). Uses an escape keyword for variables in output called "variable" followed by variable name.

3) If... else conditional module: Maps all the if, else, and else if statements by the user to the conditional statement structure in c. Has support for giving multiple relational operators in the condition.

4) For Loop module: Takes in multiple inputs from the user for all different types of implementations possible. Can have empty declarations and can handle increment and decrement of iterators easily. It handles declaration of the iterator if needed, or uses an iterator declared in the previous scope.

5) While Loop module: Maps all the while statements spoken by the user and handles any variable not declared errors by some default declaration of the variables used in the while loop. Also has support for multiple relational operators. Can use the break and continue statements to exit from the loop for some condition.

IV. Results

The foremost thing which drives our results is the accuracy of the speech-to-text engine which is a third party engine. According to cxtoday [4], Google cloud speech-to-text scored a 79% accuracy in 2020. Keeping that in mind, due to the nature of the stubs approach and static mapping, the accuracy for the framework for converting speech to source code will also be 79%. Therefore, the accuracy of the conversion, given that the user does not speak anything which is alien to the program and the conversion made by the engine is accurate, will be 100%.

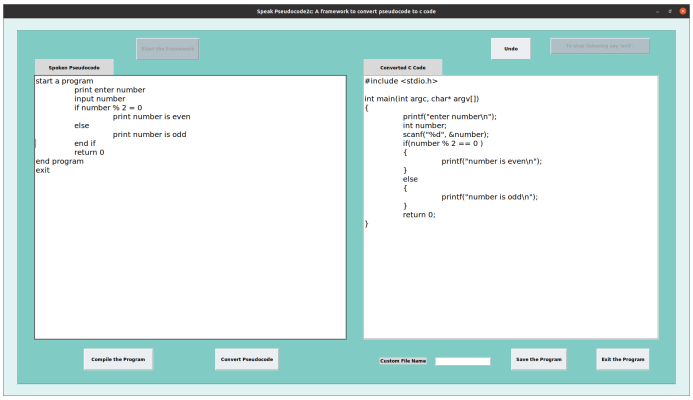

Fig. 2: This is the framework, currently converted for a program to print odd and even numbers. The left side represents the pseudocode spoken and the right side represents the converted source code in c

The framework is currently tested on a set of 30 programs chosen for beginner levels i.e. class 6^{th}, 7^{th}, and 8^{th}, 10 programs each. The accuracy of conversion on the problem set was 100% with no error. This accuracy comes with some assumptions that the user is following a certain set of rules, like giving variables meaningful names which can be spoken and recognised easily. The framework also includes tools for undoing wrong translations made by speech or otherwise, which makes it more robust to errors. Supporting keywords for compiling and saving the program are also included in the GUI and as a verbal command.

V. Conclusion and Future Work

The framework successfully converts natural language pseudocode to formatted c source code with 100% accuracy. A user can Compile and Save the generated program through the voice input. Functionality for a narration/commenting system, to convert written pseudocode directly to convert the whole pseudocode to c code at one go and an undo functionality for a robust framework and handling errors in speech. The framework is beneficial to use especially for beginners who want to learn the C language. Speak Pseudocode2c can be accessed by the users through a well formed GUI which provides real time line to line updation of voice input. The user can save the generated program with any desired file name. Even if no file name is given, it will be saved with a default name and can be accessed by the user later in time if needed. Also, when the framework is started, a set of rules are displayed to brief the user about the guidelines and limitations of the framework for smooth usage throughout.

Future work on this approach can be extending functionality to include support for strings and other data types. Presently, the framework supports C language only. The framework can be extended for other languages as well. The framework can be modified to make the process completely offline as translating and retrieving data can add latency and doing so will remove the internet dependency as well. Functionality which provides text to speech conversion maybe added. This could be helpful for the user as they can rectify if speech to text translation is correct or not.

Acknowledgment

I would like to express my gratitude to Prof. Nitin V Pujari, Department of Computer Science and Engineering, PES University, for his continuous guidance, assistance, and encouragement throughout the development for this research.

References

[1] A. Abujabal and J. Gaspers, "Neural named entity recognition from subword units," *arXiv preprint arXiv:1808.07364*, 2018.

[2] A. T. Imam and A. J. Alnsour, "The use of natural language processing approach for converting pseudo code to c# code," *Journal of Intelligent Systems*, vol. 29, no. 1, pp. 1388–1407, 2020.

[3] L. Haowen, L. Wei, and L. Yin, "An xml-based pseudocode online editing and conversion system," in *IEEE Conference Anthology*. IEEE, 2013, pp. 1–5.

[4] "How reliable is speech-to-text in 2021?" Mar 2021. [Online]. Available: https://www.cxtoday.com/speech-analytics/how-reliable-is-speech-to-text-in-2021/: :text=As per benchmarks published in,scored a slightly better 84%.

[5] P. Sirikongtham and W. Paireekreng, "Improving speech recognition using dynamic multi-pipeline api," in *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*. IEEE, 2017, pp. 1–6.

[6] T. Dirgahayu, S. N. Huda, Z. Zukhri, and C. I. Ratnasari, "Automatic translation from pseudocode to source code: A conceptual-metamodel approach," in *2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*. IEEE, 2017, pp. 122–128.