Constructing a Simple Verbal Compiler

Ahmed Laarfi, Veton Kepuska

School of Electrical Engineering and Computer Science, Florida Institute of Technology, Melbourne, USA
Email: Alaarfi2015@my.fit.edu, vkepuska@fit.edu

How to cite this paper: Laarfi, A. and Kepuska, V. (2020) Constructing a Simple Verbal Compiler. *International Journal of Intelligence Science*, **10**, 83-91.
https://doi.org/10.4236/ijis.2020.104006

Received: August 10, 2020
Accepted: September 15, 2020
Published: September 18, 2020

Abstract

The paper's purpose is to design and program the four operation-calculators that receives voice instructions and runs them as either a voice or text phase. The Calculator simulates the work of the Compiler. The paper is a practical example programmed to support that it is possible to construct a verbal Compiler.

Keywords

Speech Recognition, Artificial Intelligence, Programming Languages, Compiler Construction, Verbal Programming

1. Introduction

"We're not gonna reinvent what has already invented!"

In this stage, all the system is divided into phases. Some questions have to be practically answered through preparing programs to execute the tasks that the Compiler performs. The first question that should be asked is what this Compiler looks like? In terms of programming, the question means how to design the main menu of the Compiler and the other menus. Furthermore, the answer declares why it is called a compiler, not an application. Another question, for example, is how this system works. The solution gives details about how to merge the voice and text to run the procedures and classes. The third question is why it is a new compiler? Is it needed in the presence of this crowding of programming languages that it would not be better than them? Why confuses the programmer? The practical answer to this question will be through programming commands, from which it is clear that the used language is an old language with its known commands to any programmer. It is possible to change the writing of language commands with new commands, but this will not benefit the programmer and cost him time and effort to learn the new language. Despite the scarcity of resources on available audio programming, either in the language itself or in external libraries, only voice commands have been combined with language commands. In order to avoid the ambiguity that might lead to errors, the voice commands were restricted to specific words to be distinguished. These words are included in any language by adding them either by text file or by defining them in the main program of the Compiler.

2. The Difference between the Verbal Compiler and the App Is a Lock of Hair!

It was also discussed in a previous paper [1] that there is no need to design a new compiler but to use one of the available languages while integrating the audio libraries with it.

Here, a simple example is a simple calculator, designed and programmed to receive verbal instructions and execute them, as will be explained later.

Simple Calculator, with basic operations, is not a comprehensive example, and only some limited processes are exposed through which an attempt is made to review some essential issues in the design of compilers.

For example, to show the occurrence of an error and the type of this Error, one of the basic mathematical operations, such as division, is disrupted. Such an error is dealt with in the basic definitions or libraries defined for each operation. Thus the Error is seen and a letter or a voice message given.

In much larger applications such as databases, databases can be set up as tables. These tables are loaded into records and fields, and each table has relationships with other tables. These properties are defined by voice commands and are also filled with data. While dealing with these databases, of course, many errors occur that need to be reviewed later. Here we find that there is a need to store voice commands to enter data. These files will then assist in modifications and cancellations. Certainly, storage is not a sound, but the voice command is transformed to its written counterpart and stored in text files for programming. In the case of a Simple Calculator, the storage feature is not used, although it is taken into consideration. The non-use of this character is because there is no need to store a single command that can be restored phonetically. As for the many consecutive orders, they must be converted to their origin in the language used and stored in the program's writing file for later reference.

3. A Model of a Simple Compiler That Deals with the Voice as an Input

Here, these figures present a simplified model for Compiler that performs mathematical operations by receiving voice commands with an error rate close to zero.

The programmer can choose the design according to his vision. This design was chosen because the user is used to this form of Calculator, and there is no desire to confuse the user. These forms seem to perform a mathematical operation entering their data phonetically, which means that the matter is straightforward. On the surface, this simple design hides many program complications. The choice of a calculator as an example came so that we do not expand much by example, and such expansion has significant problems. There are many procedures and processes behind this example that will be discussed later.

Figures 1-4 represent the result of a group of programs that receive a verbal command, save it, and give the results.

4. How the Calculator Works

A critical issue must be clarified at the beginning. In the paper [1], two design options were given. One of the two options is to use text files to store the input and output data and deal with them later. The most preferred choice is to capture the sound, place it in a buffer, use it directly, or it can be stored as well.

Following the second option, which is dealing directly with sound from within the computer, has many advantages. First, the task can be performed very quickly without resorting to external communications. Secondly, contacting resources outside the machine, and thus outside the computer's memory, complicates the process of operations and increases slowness. If this connection is lost, it must be returned. Third, and most importantly, voice commands are faster to enter. For example, in this program, numbers can be entered as either a number or a set of numbers and mathematical operations according to the pronunciation.

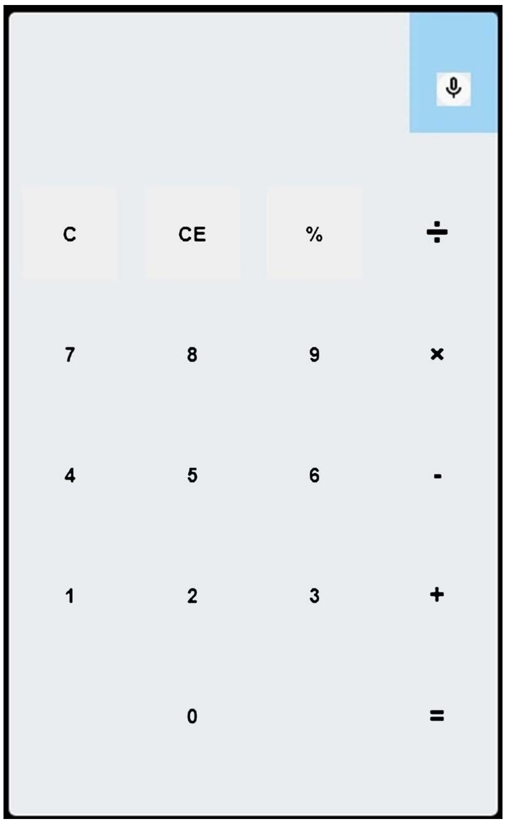


Figure 1. A verbal Calculator [A program Run].

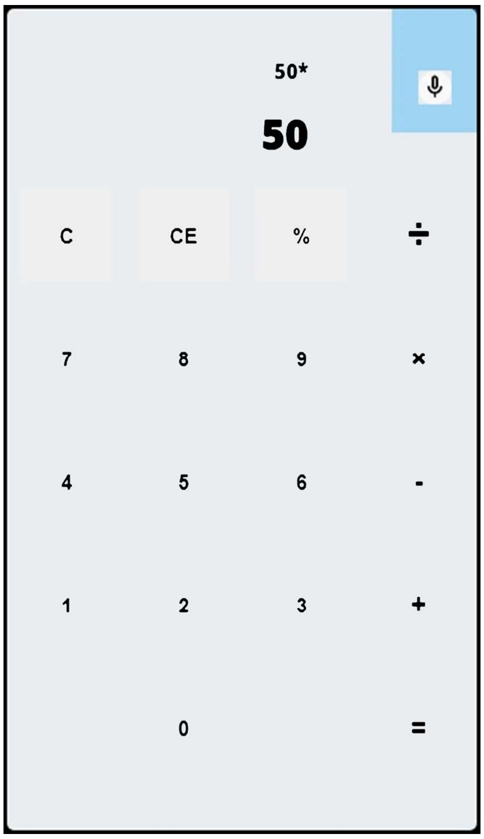


Figure 2. A verbal Calculator [A program Run].

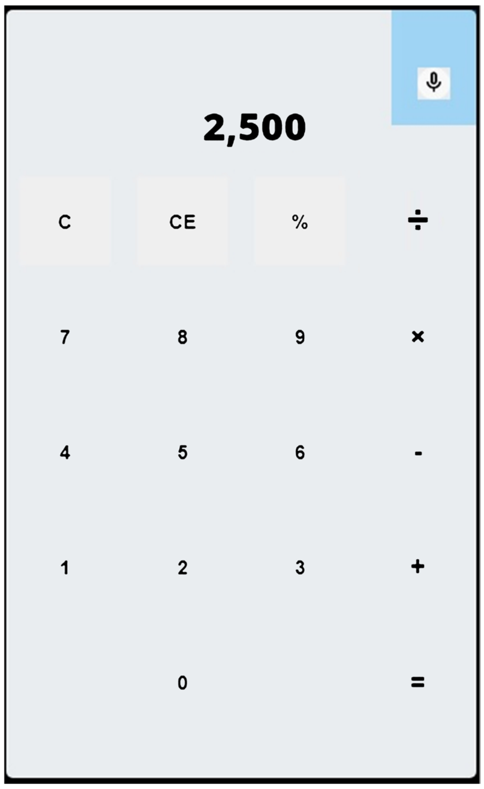


Figure 3. The result of multiplying two numbers [A program Run].

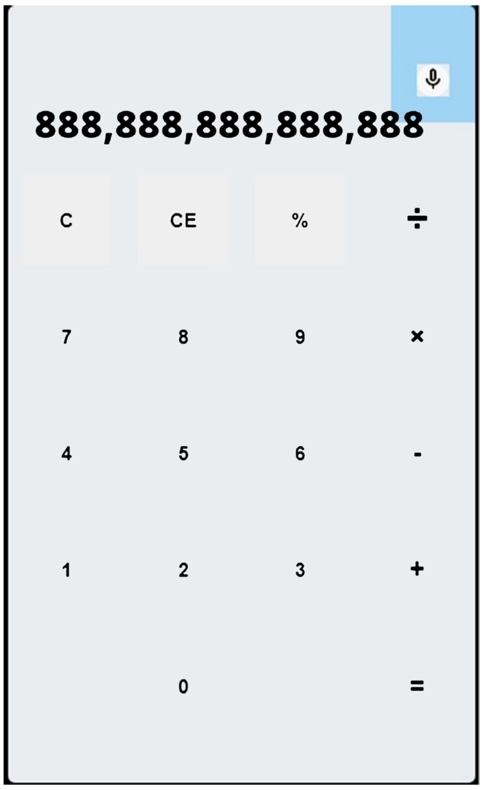


Figure 4. An error occurred [A program Run].

This Calculator simulates simple calculators, and many designs can be chosen. Some of them hide numbers and operations from the end-user interface. The user can make some adjustments to regulate the output status. The Calculator can only write the result, pronounce it, print it out, store it, or do all together.

Such processes, which take place in seconds, have high logic and processes' complexities hidden in the application

Also, it is possible to control the time that the microphone remains open and receive orders. For example, it may be turned off after a second or two if it does not capture a sound. At the same time, this short time distance can be increased in case of entering numbers and operations verbally. The percentage of the Error in the entry is almost non-existent unless it is a human error, such as not entering the amount that is supposed not to appear.

The graph above demonstrates a general flowchart of how the Calculator works. Starting from capturing the voice, controlling the time, and finally, get the results (Figure 5).

5. What Are the Processes Hidden by the Example?

The Compiler is a different programming language. One of the existing programming languages is used for computer management, through which mainframe systems, which represent the operating systems. Then there was a need to find programming languages to meet the demands of users. Companies competed to create new languages and develop the ones on the market. We have

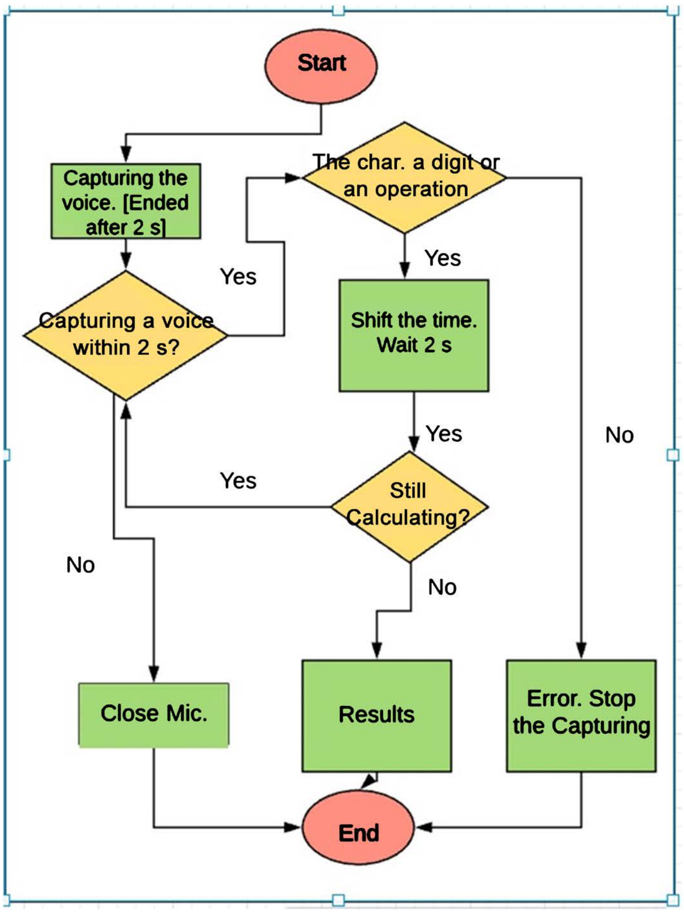

Start
Capturing the voice. [Ended after 2 s]
The char. a digit or an operation
Yes
Yes
Capturing a voice within 2 s?
Shift the time. Wait 2 s
Yes
No
Yes
Still Calculating?
No
No
Close Mic.
Results
Error. Stop the Capturing
End

Figure 5. General Idea about how the verbal Calculator works [Flowchart designed by the authors].

reached a point where the market has become saturated with languages capable of programming (everything). Creating a programming language requires a great effort and consumes time and energy and unnecessary in the presence of this crowd of programming languages and applications that have macro features, which makes them almost (open source). With the development of operating systems such that sound has become an indispensable programmatic component, many applications in the sound field have emerged, providing significant assistance to the user. Moreover, programming languages are now opened with a voice command, and voice commands execute all menu instructions. Furthermore, it was possible to write the code with the direct voice command and store what was written. We have all the supportive features and keep creating a compiler (programming language) that we control from A to Z by sound. This model is a simple attempt without being overwhelmed by the details of the details to invent such languages. If a programming language that deals with databases is chosen, the model will be very complicated as far as the programming language focuses on the simulation. The Calculator of four basic operations is a good model that briefly meets most of the programming levels that we will show. I deliberately used a well-known programming language to make a (new) programming language, and I did not change anything in the programming language that I used to produce the programming language in order not to add anything new with simple capabilities.

In contrast, the goal is to employ sound in programming. The same libraries were used with the same (Token) of the language. The addition is a simple modification in the form of a sequence of steps for the Compiler so that we added in parallel the audio libraries for the stages of the Compiler.

6. The Programming Languages from Past to Present

6.1. The Compiler, a Programming Language

At the beginning of the invention of the computer, it was the giant machine with limited capabilities. The hardware was not fit for anyone who wanted to use it. After several developments, the hardware became progressively irreproachable with software and operating systems. Now the hardware and software are advancing very quickly.

One of the most important uses of computers was programming languages. Every programming language is a compiler. Programming languages have evolved and diversified to include all aspects. On the other hand, accompanied by significant developments in the hardware and a large number of audio applications appeared. The emergence of these audio applications was not accompanied by programming languages that support voice commands. An attempt was made to find a programming language based on the voice command, not the scripture.

6.2. Why Do We Look at a Calculator as a Compiler?

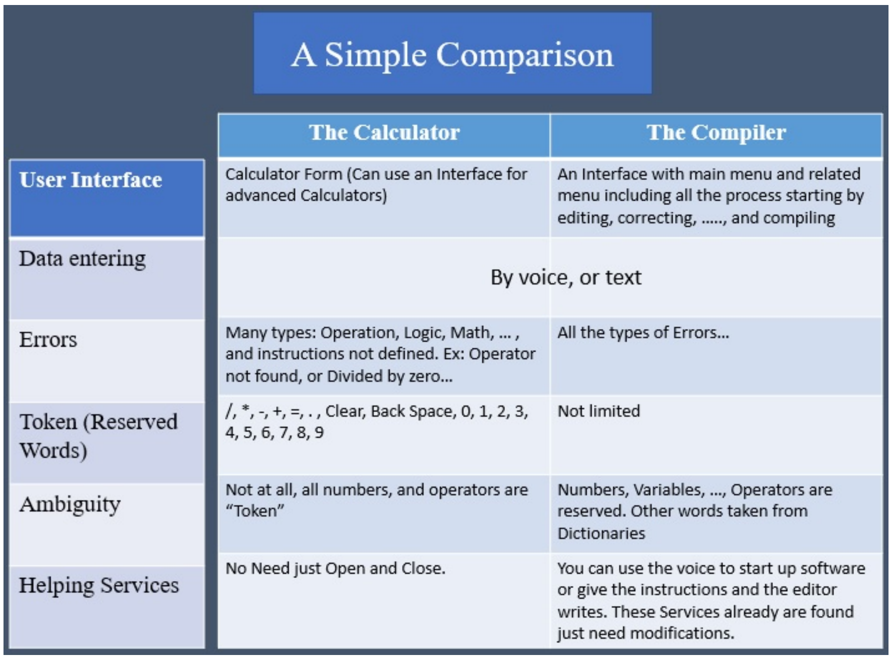
The source code is entered into the programming language used, and from it, we obtain the required program after a series of processes from the Compiler.

The Calculator is always a program. Here it was taken as a prototype for a simplified compiler. A user interface can be designed to make the programming easier; in this model, the user interface is the Calculator. In general, while creating a compiler, it should have a main menu that had all the operations in other branched menus, starting from scratch when the user is asked to choose if that program is new or saved before, passing through the programming till the run.

By compiling the program, some errors interrupt this process. They are different types of Errors, like miswriting, logic, enter instruction not found in the language, and calculation errors like the division by zero. These errors should be classified according to the type, and given numbers indicate on the Error and the error types [1]-[10] and [11] (**Figure 6**).

7. Conclusion

Since the simplified Calculator operates without problems, the mainframe can be expanded. To create a program that deals with databases with voice commands, database tables must first be analyzed and designed. Mostly they are rows called records, and a column called fields. Design the appropriate number of tables



A Simple Comparison

	The Calculator	The Compiler
User Interface	Calculator Form (Can use an Interface for advanced Calculators)	An Interface with main menu and related menu including all the process starting by editing, correcting, ….., and compiling
Data entering	By voice, or text	
Errors	Many types: Operation, Logic, Math, … , and instructions not defined. Ex: Operator not found, or Divided by zero…	All the types of Errors…
Token (Reserved Words)	/, *, -, +, =, . , Clear, Back Space, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9	Not limited
Ambiguity	Not at all, all numbers, and operators are "Token"	Numbers, Variables, …, Operators are reserved. Other words taken from Dictionaries
Helping Services	No Need just Open and Close.	You can use the voice to start up software or give the instructions and the editor writes. These Services already are found just need modifications.

Figure 6. Our Calculator is a compiler [Prepared by the Authors].

with no redundancy. These tables could later and over the years, be gigantic databases. Databases are dealt with from within a group of programs that form the whole system. This transaction is demonstrated here by the form presented. The Calculator is a good and straightforward model for constructing a verbal Compiler. No need to create a new language with its complexity, but one of the available programming languages can be used to develop the verbal Compiler. The same reserved words and symbols, instructions, logic, syntax, and library are used. The technic is how to enter the voice library to this language to act in parallel with the other Compiler stages.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Laarfi, A. and Kepuska, V. (2020) Implementation of a Verbal Compiler: The Need to Develop Audio Language to Keep Pace with Rapid Development Becomes a Necessity. *Global Journal of Human-Social Science: G Linguistics & Education*, **20**. https://www.researchgate.net/publication/339587476_Implementation_of_a_verbal_Compiler_The_need_to_develop_audio_language_to_keep_pace_with_rapid_development_becomes_a_necessity_Great_achievements_begin_with_small_dreams https://doi.org/10.34257/GJHSSGVOL20IS4PG1

[2] Pirkle, W. (2019) Designing Audio Effect Plugins in C++2nd Edition. Routledge Taylor and Friends Group. https://doi.org/10.4324/9780429490248

[3] Aho, A., Seth, R. and Ullman, J.D. (2014) Compilers: Principles, Techniques, and Tools. Pearson Education Limited.

[4] Pirkle, W. (2013) Designing Audio Effect Plug-Ins in C++: With Digital Audio Sig-

nal Processing Theory. Focal Press, Taylor & Francis Groups. https://doi.org/10.4324/9780203573310

[5] Reis, A.J.D. (2012) Compiler Construction: Using Java, JavaCC, and YaCC. *IEEE*.

[6] Boulanger, R. and Lazzarini, V. (2011) The Audio Programming Book (The MIT Press). Massachusetts Institute of Technology.

[7] Mac, R. (2009) Writing Compilers and Interpreters: A Software Engineering Approach. Wiley Publishing, Inc.

[8] Wells, M.A. (2005) Algorithms, Data Structures, and Problem Solving with C++. Ebook.

[9] Becchetti, C. and Ricotti, K. (1999) Speech Recognition: Theory and C++ Implementation. John Wiley & Sons Ltd.

[10] Fischer, C.N. and Le Blank Jr., R.J. (1991) Creating a Compiler WITH C.

[11] Holub, A.I. (1990) Compiler Design in C. Prentice Hall Englewood Cliffs, New Jersey.