**User Manual**

Key Words

The voice commands listed below are reserved for use by the speech recognition engine and perform translation

* Include: Used to include the required library files. Further accepts 3 voice commands to specify the type of the library to be included.
  + Standard- Includes a set of most commonly used libraries as predefined by the user.
  + System- Includes a set of system libraries which the user can spell out.
  + User Defined- Includes a set of user defined libraries which can be spelled out by the user.

A sample input may be of the form: <Include> <Standard> …<End>

* Define: This lets users declare preprocessor directives at the start of the program. Definitions are automatically prefixed with a #define directive.
* Declare: This mode lets users declare variables of different types using the following commands.
  + Integer
  + Float
  + Double
  + Long
  + Character
  + Pointers
  + List (Arrays)

A sample input may be of the form: <Datatype> <Variable Name> <Value><End>

* Structure: This mode lets users declare a structure and its corresponding elements. Automatically adds a typedef to ease the process of referencing the struct after declaration. Declarations do not accept values when invoked while defining a struct.

Input should be of the form <Structure> <Structure Name> <Variable Declarations> <End>

* Print: Allows printing a message or variables on screen as defined by the user.

Input should be of the form: <Print> <Content> <End>

* Input: Acts as a wrapper around the scanf command to accept one or more inputs from the stdin.

Input should be of the form: <Input> { <Data Type> <Variable Name>…} <End>

* Control: This provides easy access to the most commonly used control structures in the C language. This includes Conditionals, For loops, While loops and Do While Loops. Each of these is invoked as defined below.
  + If…Else: Allows definition of either a single If block or an If block followed by one or more Else blocks.

Input should be of the form <If> <Condition> <Executable Block><End>

* + For: Allows definition of for loops. Takes the condition as defined by the user and the block of code to be executed when the condition is matched.

Input should be of the form: <For> <Condition> <Executable Block> <End>

* + While: Allows definition of While loops which initially reads the exit condition and then proceeds to read the block of code to be executed when the condition is not matched.

Input should be of the form: <While> <Condition> <Executable Block> <End>

* + Do…While: This defines a standard Do While loop which reads the block of executable code before the looping condition.

Typical Input is of the form: <Do> <Executable Block> <Condition> <End>

* Function: This command allows users to either declare a new function or call a function which has already been defined.
  + Declare: Allows declaration of a new function. Splits the process into multiple parts to ease the process of defining the function.
    - Reads the return type of the function which can either be one of the built-in C types or user defined type. Built in types can be invoked by a single command while user defined types have to be spelled out.
    - Reads the Name of the function as defined by the user.
    - Reads the parameters of the function, if any.
    - Reads the actual block of commands the function performs. This again allows declarations, function calls, I/O statements and variable value changes as defined earlier.

Typical Input is of the form:

<Function> <Declare> <Return Type> <Function Name> <Parameter List> <Function Body> <Return Value><End>

* + Call: Allows a user to call a defined function. Reads the name of the function followed by parameters if any.

Typical Input is of the form: <Function> <Call><Function Name><Parameter List>

* Spell: This mode hands control over to the user to define commands he chooses. This works by reading the first character of each word uttered by the user. Calls the alphabetRecognize() function (described below) so as to let the user define any nonstandard code segments.

Typical Input is of the form : <Spell> <Code Segment><End>

* End: Exits out of the currently executing submodule.
* Exit: Exits out of the main module.

Underlying Functions

A brief description of each of the auxillary functions used is listed below:

These functions mostly accept no arguments unless indicated and return a string containing equivalent C code.

* voiceInclude(): Provides Support for including library files.
* voiceMacro(): Allows the declaration of Macros automatically prefixed with a #include directive.
* voiceDeclare(): Supports Declaration of new Variables and optionally defines their values. Takes in a Boolean value to indicate whether or not to support initialization of the declared variable.
* voiceprint(): Allows messages to be printed on screen.
* voiceInput(): Allows inputs to be read in via the scanf function. Reads the types of the inputs to be read in and the names of the variables being mapped.
* voiceStructure(): Allows declaration of a new struct by invoking the voiceDeclare function as many times as necessary.
* voiceBlock(): This allows a user to define a block of executable code by invoking any number of new declarations, control structures, input or output statements by iterating over the individual functions defined for each of those actions.
* voiceFunction() : Allows declaration of new functions or calls to functions already defined. In the case of a declaration, it reads a return type, function name, parameter list, if any, uses voiceBlock() to get the function body and finally reads the return type if relevant.

In the case of a function call, it reads the function name and a list of parameters, if any.

* alphabetRecognize(): A standard method defined to read on an alphabet by alphabet basis by taking only the first character of the input string into account. It also defines a few reserved keywords for easier access to special characters from within the function. These are:
  + Term: Adds a semicolon.
  + Space: Adds a blank Space.
  + Delete: Removes previous character.
  + Return: Adds a newline character.
  + Character: Calls the more comprehensive characterRecognize() function which allows the declaration of standard and special characters.
  + Number: Calls the numberRecognize() method which allows the declaration of numbers by converting their textual representations into numeric constants.
* characterRecognize() : This maps words to a list of standard and special characters. Multiple words have been mapped to the same character constant to account for different usage patterns and word similarities.
* numberRecognize(): Maps strings representing numbers to their numeric forms. Accepts a string argument to differentiate between integer and floating point representations. The “point” keyword which maps to the decimal point remains inactive when defining integer constants.
* voiceControlStructures(): Allows the definition of generic C control structures. Invokes the voiceBlock command to read a block of executable code for each of the structures.

Example Program

A simple Hello World program would require the following stream of input strings from the user according to the specifications listed above.

Include

Standard

End (Exits Library Files Declarations)

Function

Declare

Void (Return Type)

M (Any word that starts with an M)

A (Any word that starts with an A)

I (Any word that starts with an I)

N (Any word that starts with an N)

End (Exits Function Name Definition)

Void (Parameters)

Print

Hello

World

End (Exits message definition for printing)

End (Exits Function Definition)

Exit (Exits Main Loop)

Output Program

#include<stdio.h>

#include<conio.h>

#include<math.h>

void main()

{

printf("Hello World");

}

Keystroke Comparison

* Typed Program: 85 Characters
* Voice Input: 98 Characters (Assuming 4 letter words which translates to 18 words of which 5 happen to be either End or the final Exit)
* Efficiency of the model increases as the usage of inbuilt features increases. Efficiency takes a hit with each call to user defined code segments.

Next Steps

* Implementing a text editor possibly using the Java Swing interface to read and edit files.
* Expanding inbuilt feature set. Possibly an edit mode dedicated to text processing and an Expression mode for invoking mathematical and logical operations on operands.
* Plugging in the actual speech recognition system and passing the strings recognized as inputs to the system.