1. Introduction

The project aim is to build an efficient method or function searching tool that can be used to search user defined methods in source codes written C programming languages, to rank them based on user input and to provide key searching and replacing option. Most of the source codes in this era of computing are written hugely by programmers and this tool will help us to find special user defined methods or functions in a set of source codes and it also list them in a rank according to the priority defined by users.

2. Background Study

2.1. User Defined Functions

Methods or functions are the building blocks of most of the programming languages. They are used to make a block or segment of codes reusable. There are two types of functions or methods. The first type is called user defined and the other is called built in. In our project I have only dealed with user defined functions or methods. These types of methods or functions are usually written for implementing a special task in source code and named by the programmer according to the task they are performing. So if it is possible to detect all of the user defined methods or functions of large size source code then it will be much easier to understand the source code's performance, complexity, input, output etc.

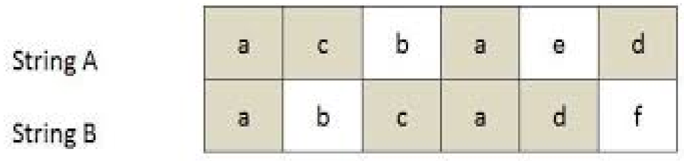
Both function in structured programming and method in object oriented programming share a common format. Usually a function returns a special value when the implementation is done and required task is performed. This particular value is called return value. A function must has return type to return a value. If a function does not return a value then the return type is void. Some values are also passed into a function to finish the implementation written inside of it that starts and ends with curly braces and these values are called parameters. If no parameter is needed then this part also can be void. Usually camel case notation is used to name a function. In structured programming a function prototype is required to write a function inside of source code that helps much to detect the function. In this project parameters and return type will also be shown to the user with function name according to their syntactical searching result.

2.2. File Stream

There various types of streams athat are available in c++ programming language. Among all of these file stream is used to take input from specific file rather than keyboard. File stream has been used to parse a provided source code contained file line by line and for string checking.

2.3. Longest Common Subsequence

A subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements. Longest common subsequence (*LCS*) of 2 sequences is a subsequence, with maximal length, which is common to both the sequences. As example LCS for sequences “abcdgh” and “aedfhr” is “adh” of length 3. Longest common subsequence has been implemented into this project to check whether a provided search key get partialy matched with any of the function of the list.



String A	a	c	b	a	e	d
String B	a	b	c	a	d	f

Fig: Example of longest common subsequence.

2.4.Suffix Trie

Suffix trie is a space efficient data structure that can be used to store all possible suffixes of a string and process strings that allows many kinds of queries to be answered quickly. It is possible to match a substring of a string in a liner time using suffix trie. Into this project suffix trie has been implemented to get the length of the matched part of a substring.

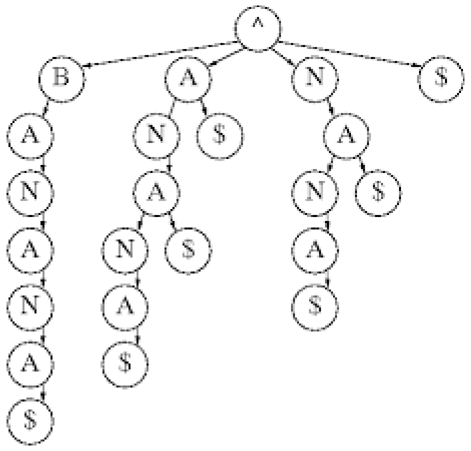


Fig: Example of a suffix trie.

2.5. Term Frequency and Inverse Domain Frequency

In information retrieval , tf–idf, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval, text mining, and user modeling. The tf-idf value increases proportionally to the number of times a word appears in the document, but is often offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. Nowadays, tf-idf is one of the most popular term-weighting schemes. Variations of the tf–idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. tf–idf can be successfully used for key words filtering in various subject fields including text summerization and classification. Term frequency and inverse domain frequency have been implemented to gain the value of a search key form inside of a function or method body.

Term Frequency

Term frequency is a ratio between two integer number, the first one is the frequency of a key word that has been appeared in a text and the number of the total words in that text.

Tf =(frequency of that key word)/(number of total word).

Inverse Domain Frequency

Inverse domain frequency is the natural logarithm of a ratio between the number of total words in a text document and frequency of the key word.

Idf =log(1/Tf)

and finally Tf-Idf =Tf*Idf

$$\mathbf{tfidf}_{i,j} = \mathbf{tf}_{i,j} \times \log\left(\frac{\mathbf{N}}{\mathbf{df}_i}\right)$$

$\mathbf{tf}_{i,j}$ = total number of occurences of i in j

$\mathbf{df}_i$ = total number of documents (speeches) containing i

$\mathbf{N}$ = total number of documents (speeches)

Fig: Equation of Tf-Idf

2.6. Spelling Checking Algorithm

In computing, a spell checker (or spell check) is an application program that flags words in a document that may not be spelled correctly. Spell checkers may be stand-alone, capable of operating on a block of text, or as part of a larger application, such as a word processor, email client, electronic dictionary, or search engine. Spelling checking algorithm has been implemented to check the spelling of a search key before starting main ranking process.

Peter norvig spelling checking one of the most common, initial and powerful algorithm to check correct spelling. This algorithm can generate a list of correct spelled word from a collection against a key input. Performance of this is this algorithm is done by deleting, inserting, replacing and transposing a keyword by single character and comparing them against a list of correct words and then picking them up .

3. Broad Domain

It has been focused to develop the project for sources codes written in C programming language. It also may work for languages whose syntax format is similar to C language as like C++ , java etc.

4. Challenges And Timeline

Activity	January	February	March	April
1. Raw coding	■	■	■	■
2.Longest common subsequence	■			
3.Suffix trie		■		
4.Term frequency and Inverse domain frequency		■	■	
5.Spelling Checking algorithm			■	■
6.Searching and replacing				■
7.Function call matrix construction				■

5. Dependencies

1. g++ 11.01 ubuntu~16.10 compiler
2. Codeblocks-16.01
3. c++ 11.01 programming language.
4. Linux (Ubuntu 16.10) operating system.

6. Methodology

Step-1 (Generate Active codes)

It is considered that commenting in source code during writing a code is a good practice for programmers. But comments in source code are obstacle for this projects because unused source codes can be commented out and it is difficult to identify unused codes without removing them. So at the beginning of this project all comments from a source code have been removed. At first single line comments have been removed and a new file has been created that still contains multi line comments. Then another new source code has been generated from the second one removing multi line comments using flaging technique. The final text file contains only active lines.

Step-2 (Collect All Functions from Current Source Code File)

After removing comments from the current text file and picking up all the methods that have been collected before, all the methods that are available into the current text file have been collected and have been inserted into the list using raw coding and condition checking. All of them have been saved into individual text file alphabetically for optimizing searching.

Step-3 (Exact Matching)

After getting the search key as input from the user at first exact matching part will have to be done. The provided search key is matched brute forcely against the collection. List elements that are matched are taken into another collection which is a short form of the main list and is given the highest priority which will take them to the top of suggestion list.

Step-4 (Implementation of Suffix Trie)

Then that collection of methods is being represented in a suffix trie representation. Suffix trie is a rooted trie used for storing associative arrays where keys are usually strings. Edges are often labeled by individual symbols. Then common prefixes are factorized. Each node of the trie is associated with a of a string of the set of strings: concatenation of the labels of the path from the root to the node. The root is associated with the empty string. Strings of the set are stored in terminal nodes (leaves) but not in internal nodes.

Step-5 (Traversing Suffix trie)

Then according to the user’s keyword, a recursive process traverse the suffix trie, and get back the destined file/folder’s path reference. It traverses upto the end character of the keyword. And traversing process ends in one/multiple leave/s. Each leaf contains a string which indicates the directory. Finally it output that string.

Step-6 (Implementation of Term Frequency and Inverse Domain Frequency)

Term frequency and inverse domain frequency are one of the common and powerful ranking formula for text mining, information retrieval. They represent how important a pattern or search key into a text. After parsing all of the line of a method that were saved before into text file every line of that file is splited into an array of words. Then each element of the array is matched with the search key to count the frequency of the search key and total words of the text file. After calculating term frequency and inverse domain frequency values of both are added to the weight of each method.

Step-7 (Implementation of Spelling Checking Algorithm)

Peter Norvig spelling checking algorithm performs three operations as insertion, deletion, replacement recursively with a edit distance to find out some exact correct form of a provided user input from a list. This algorithm is implemented here to find out method names that are intended to find are miss spelled shortly. For avoiding complexity both time and memory up to two edit distance have been used.

Step-8 (Partial Matching)

After implementations of all required algorithm are done the partial matching will get started. Partial matching will rank those methods that were not found during exact matching part by assigning a weighting value generated by those algorithms that were implemented before.

Step-9 (Key Searching and Replacing)

Finally this project will require another searching keyword from user to replace inside into the body of a selected function.

Step-10 (Function Call Matrix Generating)

Function calling matrix is simple matrix which every element is either 0 or 1 and it contains the information of that a function that has been called by other functions. If a function calls another function then their corrosponding row and column is 1 otherwise it is 0. It is possible to gain the overall diagram of a full source by taking a view on the function calling matrix.

7. Achievements

7.1. Technical Achievements

1. Raw coding.
2. Implementation of suffix trie.
3. Implementation of various algorithms specially searching and replacing.
4. Application of recursion and pointer.
5. Application of file pointer and stream.

7.2. Personal Achievements

1. Experience in handling large sized source code.
2. Increasing own coding performance.
3. Experience in implementing algorithms

8.Analysis, Design and Implementation

The raw code of this project is splitted into five modules. Module-1 is about to generating all the active user defined methods from source code. Module-2 is about to save them in different text file for each. Module-3 is about to ranking all of them based on a search key provided by user. Module-4 is about to searching and replacing another search key on a particular method or function body. And the final module is to generate the function calling matrix. All segments are dependent on one another. Modules hierarchy must be followed to run this project.

Module-1

Coding module -1 is designed based on basic structured programming knowledge. The task of this segment is to collect and store all active codes of source files all single line and multi line comments have to be removed from the code.

Module-2

The task of this module is to collect and store all active functions of source files. Then it have to parse every line of the text file to find that whether a line contains a functions or not. When all functions or methods are collected and saved into separate text file then module-2 is done.

Module-3

Coding module-3 mainly implements the ranking part of this project. A search key is asked and based on this key all functions are using exact matching, peter norvig algorithm, suffix trie length, longest common subsequence, term frequency and inverse domain frequency. After ranking all of them have been showed to user to for next steps.

Module-4

This module of this project contains searching and replacing. It requires two key string to perform this module.

Module-5

The last module of this project generates the function calling matrix of the entire source code file in a .csv extended file.

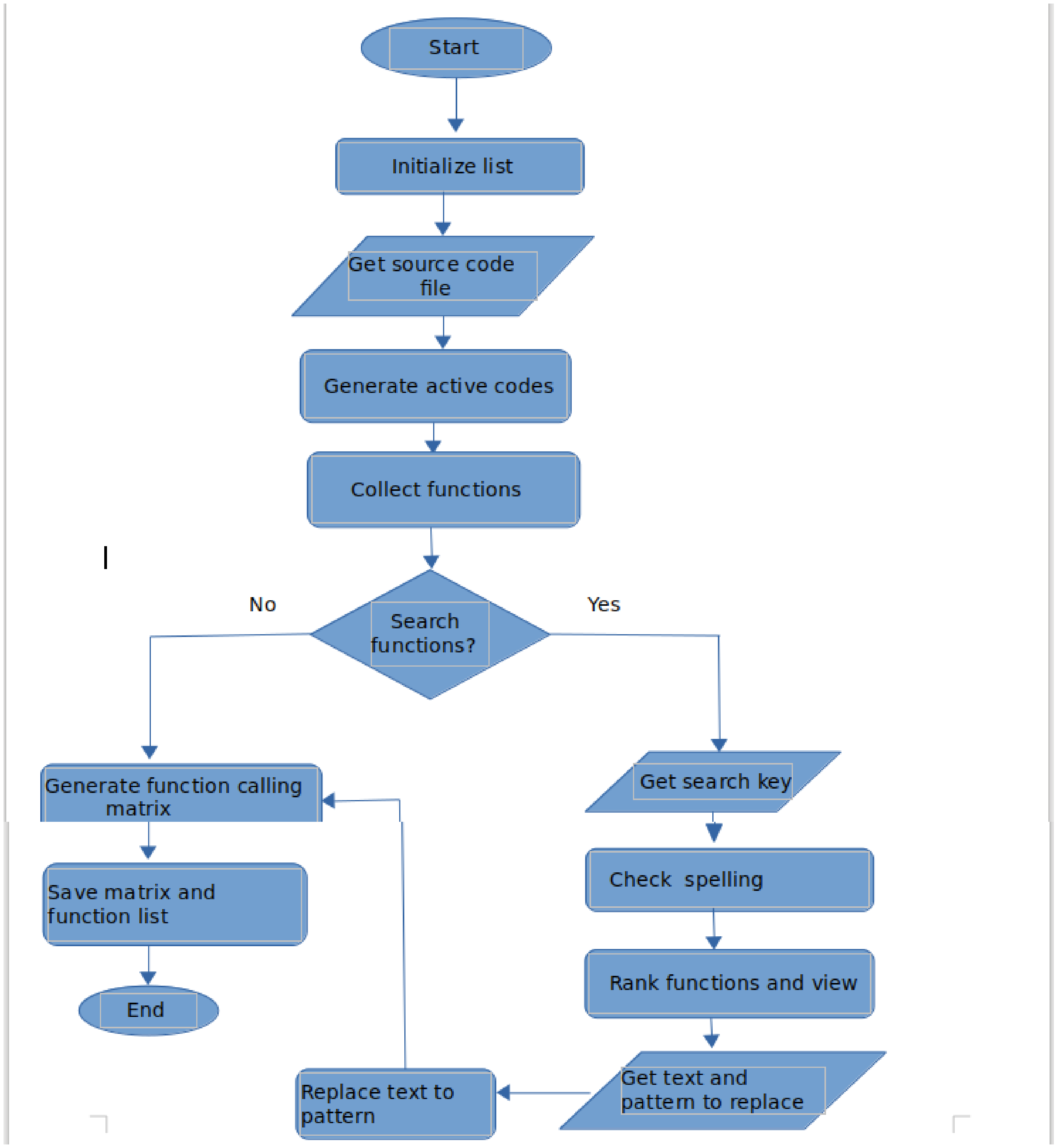
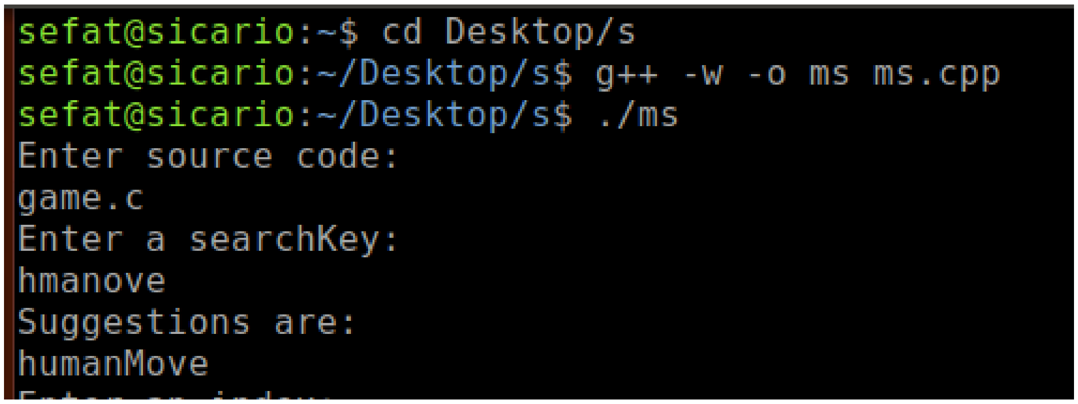


Fig: Analysis, design and implementation flow chart.

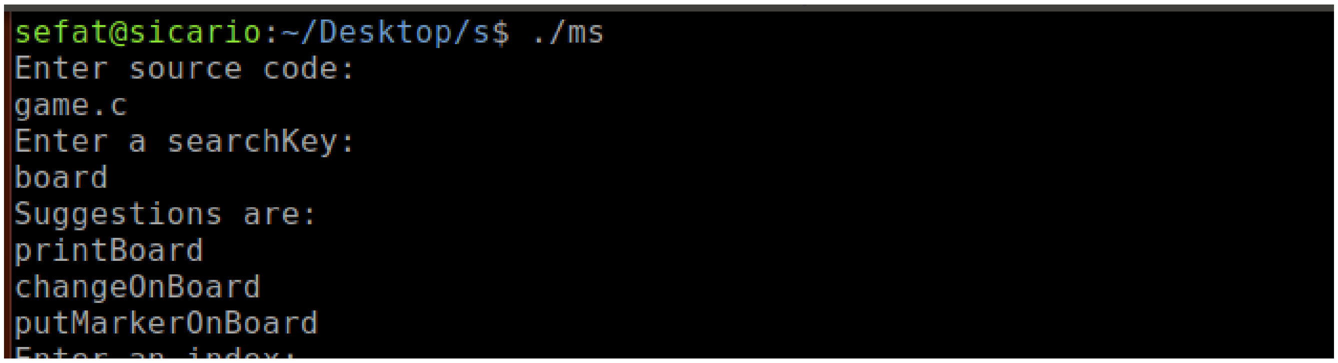
9.Program Output and Testing

Here are some examples of testing and program outputs for an input name game.c



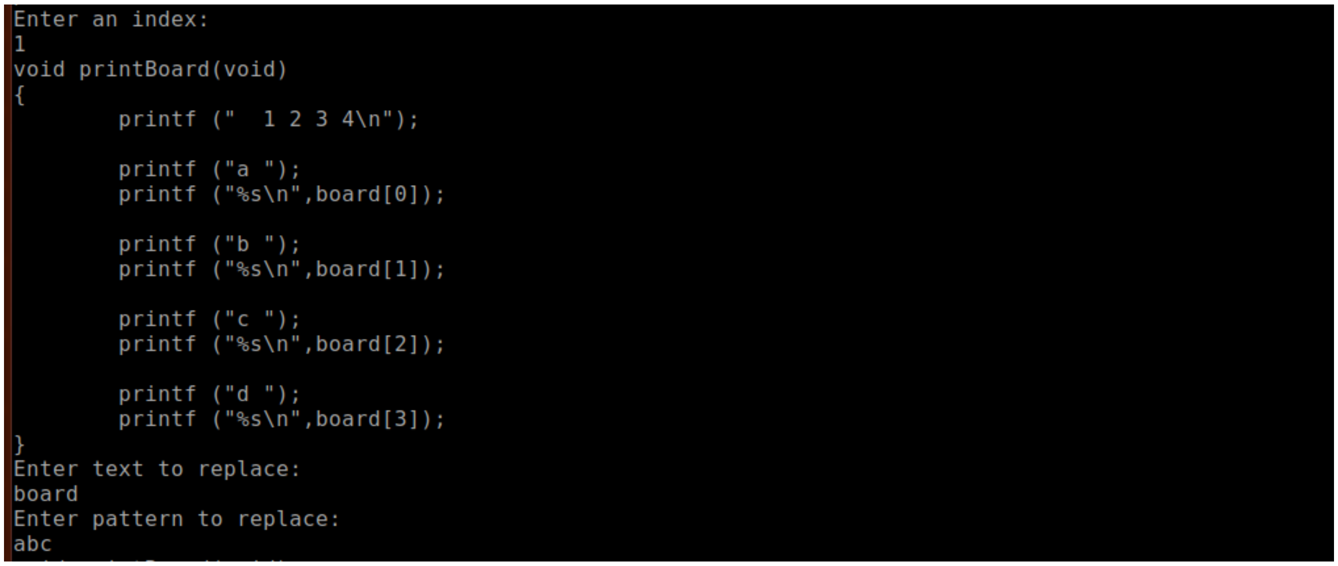
```
sefat@sicario:~$ cd Desktop/s
sefat@sicario:~/Desktop/s$ g++ -w -o ms ms.cpp
sefat@sicario:~/Desktop/s$ ./ms
Enter source code:
game.c
Enter a searchKey:
hmanove
Suggestions are:
humanMove
```

Fig: Test 1



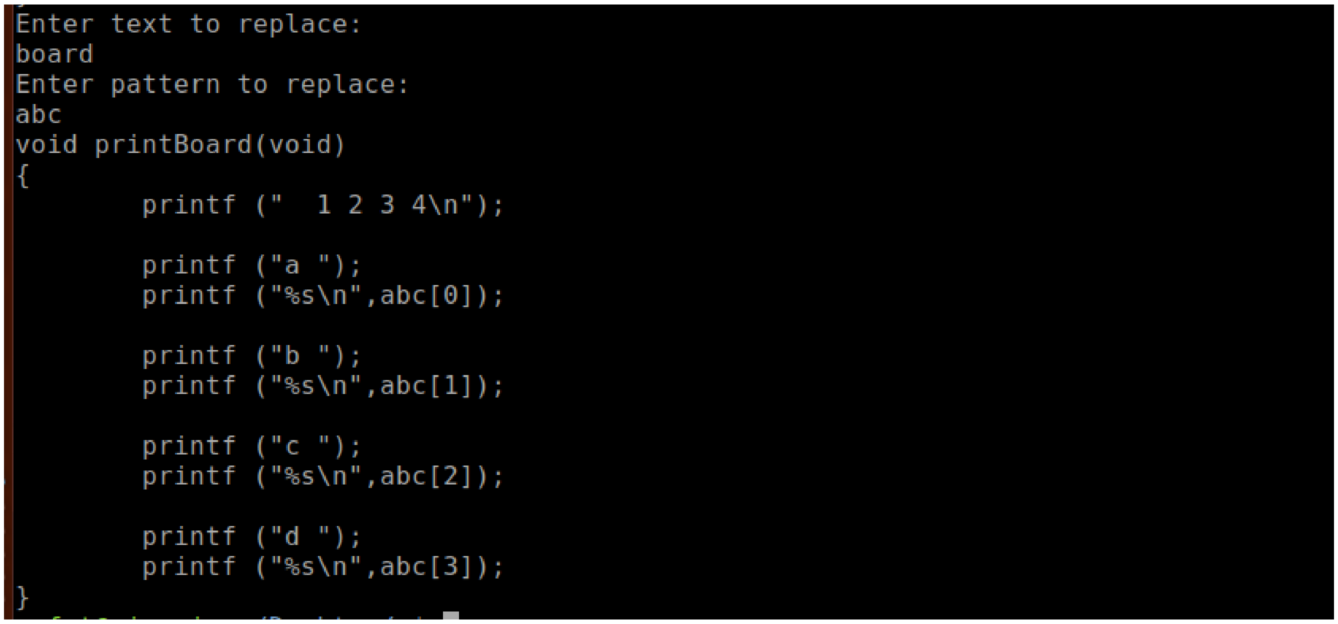
```
sefat@sicario:~/Desktop/s$ ./ms
Enter source code:
game.c
Enter a searchKey:
board
Suggestions are:
printBoard
changeOnBoard
putMarkerOnBoard
```

Fig: Test 2



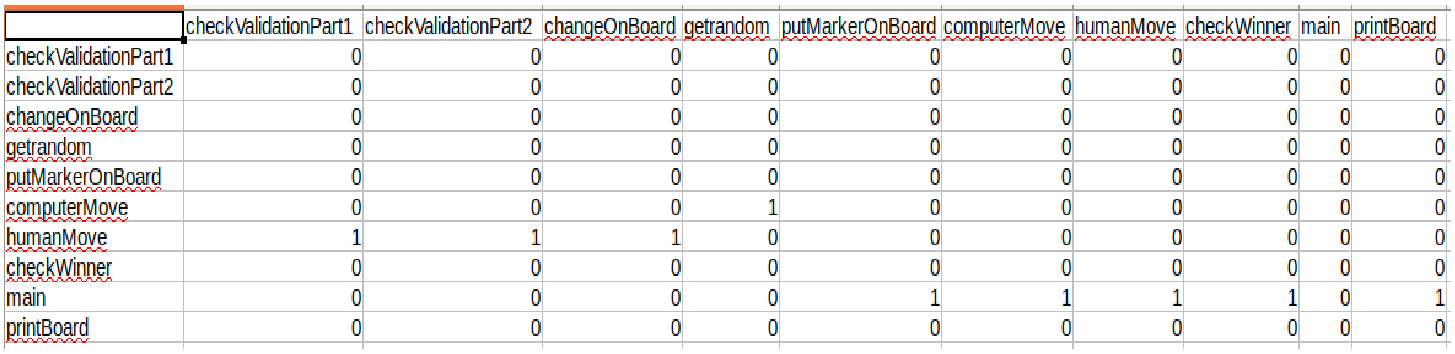
```
Enter an index:
1
void printBoard(void)
{
        printf ("  1 2 3 4\n");

        printf ("a ");
        printf ("%s\n",board[0]);

        printf ("b ");
        printf ("%s\n",board[1]);

        printf ("c ");
        printf ("%s\n",board[2]);

        printf ("d ");
        printf ("%s\n",board[3]);
}
Enter text to replace:
board
Enter pattern to replace:
abc
```

Fig: Before replacing



```
Enter text to replace:
board
Enter pattern to replace:
abc
void printBoard(void)
{
        printf ("  1 2 3 4\n");

        printf ("a ");
        printf ("%s\n",abc[0]);

        printf ("b ");
        printf ("%s\n",abc[1]);

        printf ("c ");
        printf ("%s\n",abc[2]);

        printf ("d ");
        printf ("%s\n",abc[3]);
}
```

Fig: After replacing



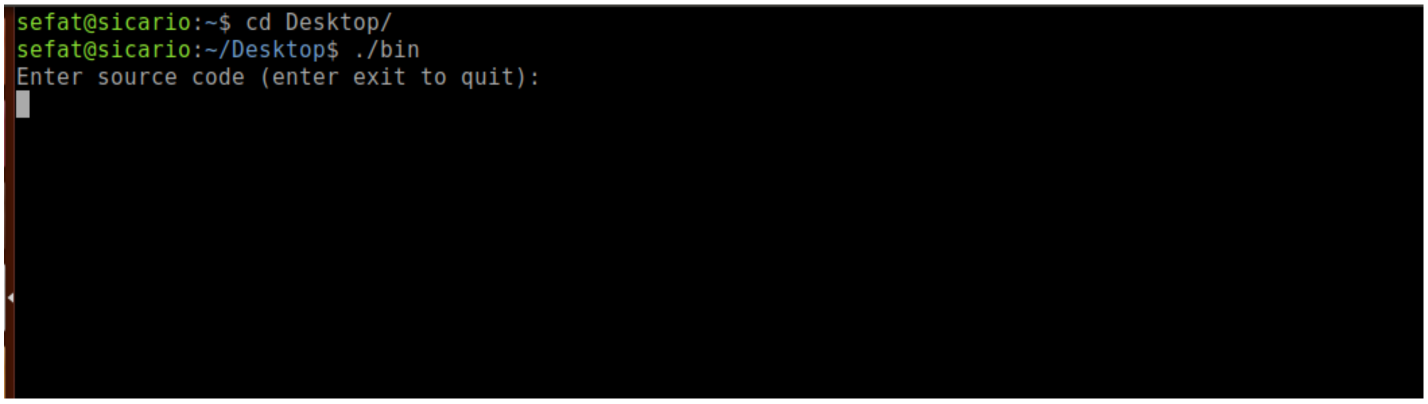
	checkValidationPart1	checkValidationPart2	changeOnBoard	getrandom	putMarkerOnBoard	computerMove	humanMove	checkWinner	main	printBoard
checkValidationPart1	0	0	0	0	0	0	0	0	0	0
checkValidationPart2	0	0	0	0	0	0	0	0	0	0
changeOnBoard	0	0	0	0	0	0	0	0	0	0
getrandom	0	0	0	0	0	0	0	0	0	0
putMarkerOnBoard	0	0	0	0	0	0	0	0	0	0
computerMove	0	0	0	1	0	0	0	0	0	0
humanMove	1	1	1	0	0	0	0	0	0	0
checkWinner	0	0	0	0	0	0	0	0	0	0
main	0	0	0	0	1	1	1	1	0	1
printBoard	0	0	0	0	0	0	0	0	0	0

Fig: Function call matrix

10. User Manual

At first user have to place the binary file and source codes which will be provided as input into the same directory. After then opening terminal the binary file must have be to run.

User/Operator presses keyword (here keyword means full function/method name or half name) in a linux terminal and according to this keyword this tool shows all the method list in a millisecond. By which user can easily find his needed function/method. The more specific the keyword is, the less amount of function/method list will be.



```
sefat@sicario:~$ cd Desktop/
sefat@sicario:~/Desktop$ ./bin
Enter source code (enter exit to quit):
```

Fig: Input terminal

11. Conclusion And Future Scope

Although this project have been done before many times by others, I have performed this project to improve my own coding skill, thinking ability and large source code handling ability. Although I have stopped here with this project but I hope that I will continue it in future to add more feature.

12. References

1. http://www.cprogramming.com/tutorial/lesson4.html
date: 22/04/2017 time:10:33

2. http://www.geeksforgeeks.org/pattern-searching-set-8-suffix-tree-introduction/
date: 21/04/2017 time: 09:10

3. http://norvig.com/spell-correct.html
date: 21/04/2017 time:09:30

4. http://www.tfidf.com/
date: 20/04/2017 time:06:05