Spell Checker

DSA project

AU-IICT
-Shikhar Pandya (121050), Shashwat Sanghavi (121049)

Table of Contents

1. Brief Description of project:-

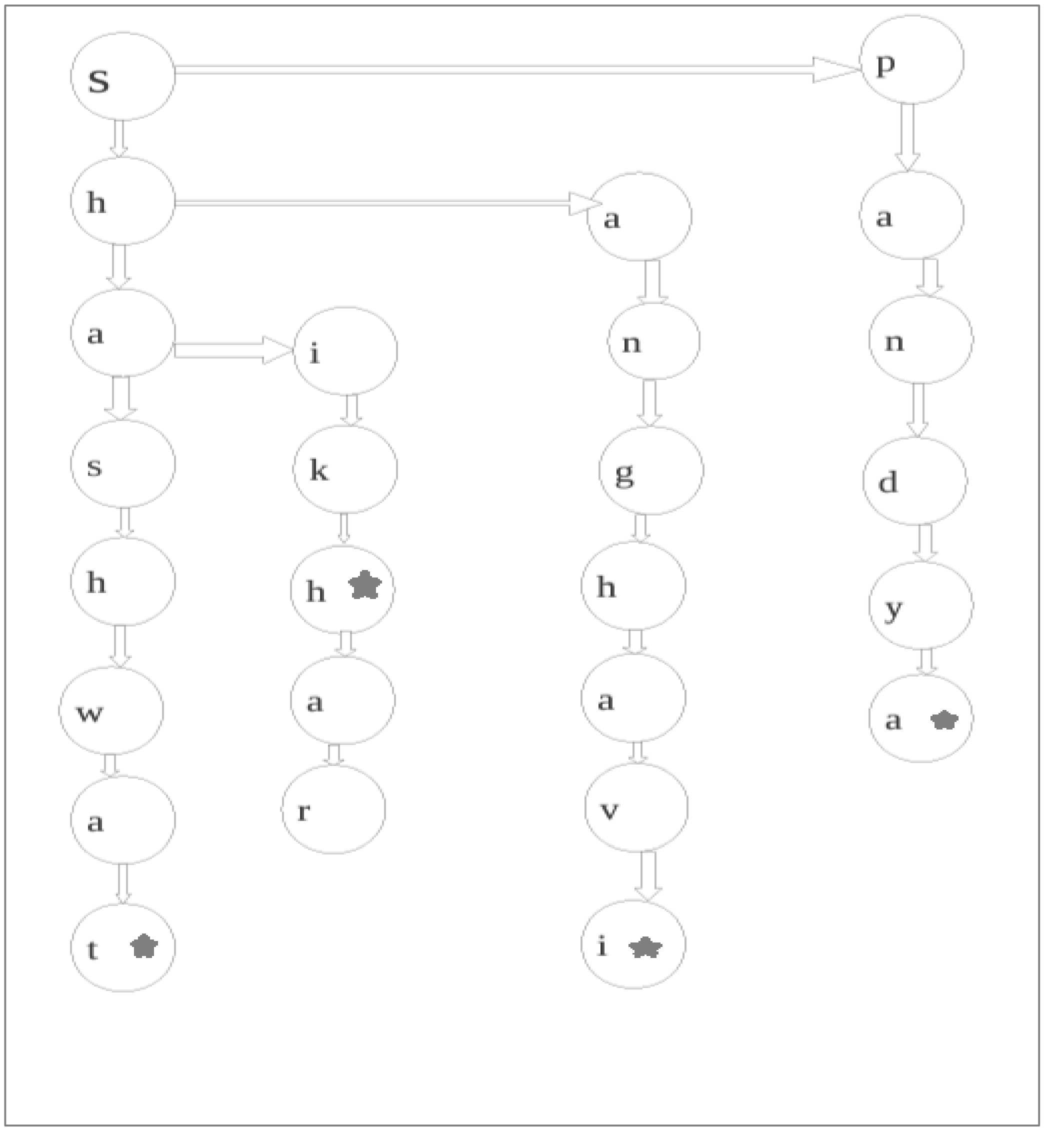
We have created a ‘Spell Checker’ for English text documents. Our program can accept a text file and scan it word by word in-order to check for spelling mistakes. It would scan the whole document and would display the words which have been misspelt. We also provide the facility to add some words to the dictionary, so that when that particular word is scanned from a text document the next time, it would not flag that word as a spelling mistake. Conversely we also provide the facility to delete the word from the dictionary. In addition to inserting and deleting a word, we also provide suggestions for the misspelt word. The suggestions are words which are similar to the misspelt word, but are present in the dictionary (meaning that they are valid English words).

2. List of Data-structures used:-

- **Trie**
- **Stack**

2.1. The ‘Trie’ data-structure:-

Trie is a multi-way tree structure, useful for storing strings. It is especially used for storing words of a dictionary. Each node in the ‘Trie’ data-structure stores a single letter of any given word. The creation of ‘Trie’ data-structure hinges on the idea that- words sharing common prefix, hang from a common node. There is also a terminator associated with every last letter of a word (to indicate that a word has been scanned). For example the words- ’shashwat’, ‘shikhar’, ‘sanghavi’ , ‘pandya’, ‘shikh’ can be stored in ‘Trie’ data-structure in the following way-



Notice that whenever a word is formed, a terminator is associated with the last character of the word (e.g. the word-'shashwat' has a terminator associated with it with the character-'t'). Also notice that the words with common prefix, hang from a common node (e.g. the words- 'shikhar' and 'shashwat'. Here the words differ from the letter – 'a' in 'shashwat' and hence these words hang from a common prefix – 'sh').

One of the primary reasons of using 'Trie' as the data structure is that it not only stores words in an efficient way, but it also reduces the time required to search a word stored in the database.

2.1.1. Why did we use *TRIE:-*

A spell checker can also be implemented using other concepts (e.g. HASH TABLES).

1. Searching a word is faster in TRIES.

2. TRIES can also be used to find words with common prefix.
3. Memory is also used efficiently as words are not stored independently (they are stored as characters and furthermore words with common prefix are not allocated separate memory).

3. List of programs and the name of their files:-

1. Header file:-
    a. spellchecker.h
2. Source files:-
    a. checkDocument() - checkDocument.c
    b. importTrie() - importTrie.c
    c. exportTrie() - exportTrie.c
    d. insertTrie() - insertTrie.c
    e. INSTALL() - INSTALL.c
    f. Main() - main.c
    g. printTrie() - printTrie.c
    h. searchTrie() - searchTrie.c
    i. stack() - stack.c
    j. autoComplete() - autoComplete.c
    k. suggestion() - suggestion.c

NOTE: click on the name of the file to open the source code.

4. Complexity:-

➢ **Storing a huge database of words (which amounts to 99,171 words to be precise)**
Storing such a huge database of words is in itself not a big task, but using an appropriate data-structure, so that memory can be used efficiently is what makes this project complex.

➢ **Storing the words onto a file and recreating the data-structure, every-time the program is run.**
Each functionality is implemented by manipulating data-structures. The user is not supposed to enter a the whole database of English words word by word, every-time he/she wants to check a text file.
The complexity is that one needs to store the data in such a way that it is easy to retrieve and easier to recreate the data-structure (after reading from the file).

➢ **Suggesting options for misspelt words (We developed our own algorithm for it).**
A good spell checker is one which not only points out mistakes but also suggests good alternatives.
There were many algorithms available on the internet which implemented ‘word-suggestion’ functionality. We studied them and got the essence of it. We devised and implemented our own algorithm using the basic idea of ‘word-suggestion’.

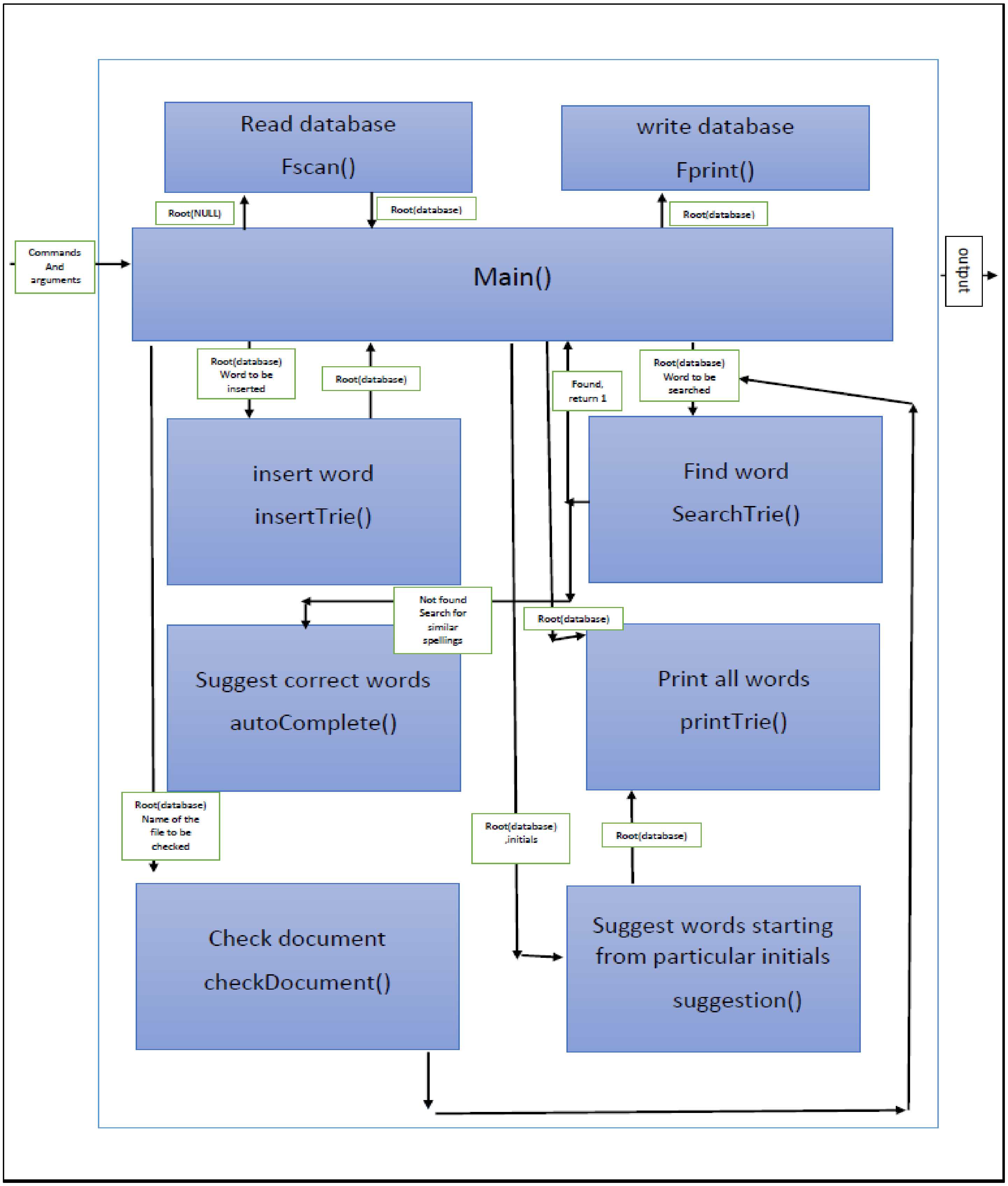
Complexity is developing our own algorithm after getting a basic know-how of how the ‘word-suggestion’ algorithms work.

- **Providing Command line interface for running the program.**
  We had visualized our program as a utility. Command line interface is necessary if one want to integrate this utility with any software.
  Complexity is- linking the command line arguments with respective functions.

- **Providing help to the user in order to run our program successfully.**
  The user may need help in order to run our program. Providing appropriate print statements can help him use the command line interface without any problem.
  Complexity is- predicting what kind of confusion the user would face.

- **Installation file**
  We needed to compile all the files together and for that we could have used the ‘make’ utility, or we can use an IDE and use the ‘project’ facility. Instead of using the above methods, we tried something new. We used the ‘system’ command to compile all the files in a single go.
  Complexity is- to come out with an ingenious idea to compile in such a fashion.
  Also while using this program we have taken care that if the binary is already there, then the program will ask the user to regenerate the file.
  Also if there is any compile time errors then the program will stop executing at that movements and errors will be printed on the terminal.

5. Program architecture:-



6. Algorithms:-

6.1. Inserting words in a Trie:-

PRE: temp=root; Word to be inserted.

POST: Pointer to the Updated database.

```
1) Check if root is null-
    1. If yes- do (until the last character of word to be inserted is reached)
        1. Store character in the 'temp' node
        2. temp=temp->firstChild
        3. Check if (last character of Input word)
            1. if yes – set 'flag' = 1
            2. else – continue
    2. else-do (until last character of word to be inserted is reached)
        1. check if - the character in input word matches with that stored in node
            1. if yes- temp = temp->firstChild
            2. check if temp is NULL
                1. if yes – do (until last character of word to be inserted is
                   reached)
                    1. Store character in 'temp' node.
                    2. Check if – last character of input word is reached-
                        1. If yes- set 'flag' = 1
                        2. else- continue
                    3. BREAK.
            3. check if last character of input word is reached-
                1. if yes- set 'flag' = 1.
                2. else- continue
            4. else-
                1. temp = temp->nextChild
                2. do (until last character of word to be inserted is reached)
                    1. store character in 'temp' node.
                    2. Temp = temp->firstChild
                    3. Check if - last character of input word is reached-
                        1. If yes- set 'flag' = 1
                        2. else- continue
                        3. BREAK.
```

6.2. Searching Words in Trie:-

PRE: temp=root; Word to be searched.

POST: '1' if found, else '0'.

```
1) Check if-Root is NULL
    1. If yes- do (until last character of word to be inserted is reached )
```

```
        1. Check if- the current letter in the word to be searched matches with that
           stored in node
            1. If yes- temp=temp->firstChild
            2. Check if-( (it is the last character of the word to be searched) AND
               (   check if -flag==1 for that node))
                1. If yes- print: Word Present
            3. Check the next character in the input word with the character
               stored in current node and goto step 1.1
            4. else –temp = temp->nextChild
            5. Check if –((it is the last character of the word to be searched) AND
               (check if -flag==1 for that node))
                1. If yes- print: Word Present
            6. check the same character in the input word and step 1.1.1
    2. else- Print: No Data-structure not created yet.
```

6.3. Writing Words onto a file:-

PRE: temp=root.

POST: Printed database in file.

```
1) Check if temp is NULL
    1. If yes- PRINT: No Data available
    2. else-
        1. Open a file to write data
        2. Store the character stored in temp (root) into the array-word at index-level
        3. Do (until level=30)
            1. Write character stored in 'word' at index-level onto the file
            2. Check if there is a nextChild of temp-
                1. If yes-push the node onto the stack.
            3. Check if (firstChild of temp is not NULL) AND (flag of temp is  ='1')  //(i.e.
               word within a word)
                1. If yes-
                    1. Write '\n' onto the file
                    2. Write all characters stored in 'word' onto the file.
            4. Check if firstChild of temp is not NULL
                1. If yes-
                    1. Assign temp with firstChild of temp
                    2. Increment level
                    3. Store the character stored in that node in 'word' at
                       index-level
                    4. Continue
                2. Else if-Top Of Stack is not equal to NULL-
                    1. Pop from stack and assign temp the popped node
                    2. Assign temp with the nextChild of temp
                    3. Store the character stored in node into-'word' at index-
                       level
                    4. Write all the characters stored in-'word' onto the file.
                3. Else if- Top of stack ==NULL
                    1. Close the file
```

6.4. Recreating the data-structure after Reading from the file:-

PRE: temp=root.

POST: Pointer to the Updated database.

```
1) Open file
2) Scan word from the file using a character array (say spell)
3) Do(until last character of spell is not EOF)
    1. Insert the word into the trie (by calling-insert function)
    2. Again scan a word from the file using character array
    3.  store the character pointed to by file pointer into the last character of array //(if
       the word scanned is not equal to the total length of the character array)
4) Insert the word into the trie (by calling insert function)
5) Close file
```

6.5. Printing words from the database:-

Same as that of writing onto the file (the only difference being-here we are printing onto the terminal)

6.6. Auto Completion :-

PRE: temp=root; Word to be searched.

```
1) Check if- temp=NULL
    1. If yes-PRINT: No data available
    2. Else- do(while counter is less than total length of the word to be searched)
        1. Check if(character stored in temp node matches with the character in the
           string(word to be searched)at the index-counter)
            1. Check if-(firstChild of temp is NULL)AND(counter has not reached till the
               total length of the word to be searched)
                1. If yes-set Emptyfirst flag to be '1'
                    1. Break from loop
            2. Assign temp1 with temp
            3. Assign temp with firstChild of temp
            4. Increment counter
            5. Continue the loop
                1. Else if-(nextChild of temp is NULL )
                    1. If yes-set EmptyNext flag to be 1
                        1. Break from loop
```

            2. Assign temp1 with temp
            3. Assign temp with nextChild of temp
            4. Continue the loop
    3. Check if (EmptyFirst flag is set to ‘1’)OR(EmptyNext flag is set to ‘1’)
        1. If yes-PRINT: Word not found
        2. Print the word onto the terminal.

6.7. Suggestion/What to suggest:-

It is same as that of ‘printTrie’. The only difference lies here is that instead of printing on the terminal we store the words in a different array (‘\n’ indicates that a word has been formed in the TRIE). Now what the suggester does is-it compares the input word (misspelt word) with each and every word in the database. And whenever it encounters a word which is close to the misspelt word, it prints that word as a suggestion.

7. Algorithm(s) and Code analysis:-

Time Complexity:-

O(K) is the time complexity of TRIE data-structure in its worst case scenario, where K is the size of the string. Worst case scenario is when the input word has no common characters

8. Limitations and Drawbacks:-

While we do provide the functionality to insert words into the database, there is an inherent flaw in it-the person might enter any arbitrary sequence of characters (which the ‘Trie’ identifies as a word). For example- the person might enter ‘asdfg’ as a word into the database. Our program cannot distinguish between an arbitrary sequence of characters and a valid English word, once that arbitrary sequence is added to the database (This basically implies that the software can be easily corrupted).

Deletion of a word can only be done by explicitly opening the text file containing the database of English words and then deleting it.

9. What more we could have done:-

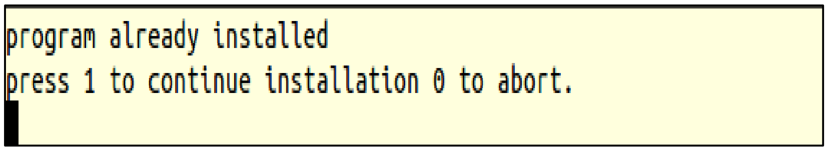
If we had a little more time, we wanted to extend our spell checker for Gujarati text documents as well.

Also as suggested by Prof. Sanjay Chaudhry, we would have liked to implement the functionality of ‘Thesaurus’, whereby we could have suggested homonyms, synonyms and antonyms for a given word.

10. Instructions to execute program

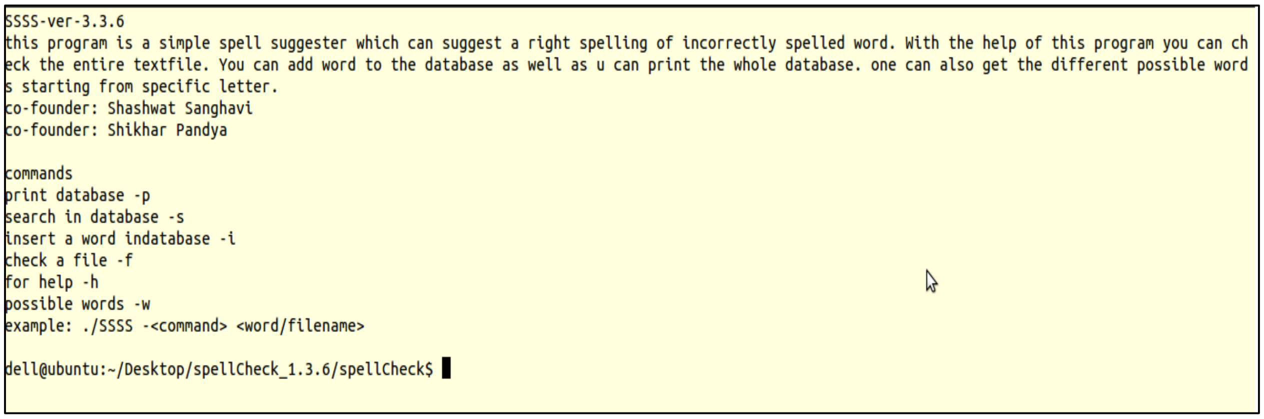
1) Compile INSTALL.c to get INSTALL.out by following command. (cc –o INSTALL INSTALL.c).
2) Now run the INSTALL by following command to compile all the files and you will get SSSS.out file. (./INSTALL)
3) Executable file of our program is ready its name is SSSS.out.
4) Since we are using command line arguments give appropriate command line argument to execute the executable file. For commands and their usage visit help page of the program by the following command. (./SSSS -h)
5) Enter your command and other arguments as shown in the help page.

11. Screenshots:-



```
program already installed
press 1 to continue installation 0 to abort.
```

SS1: Execution of install program.

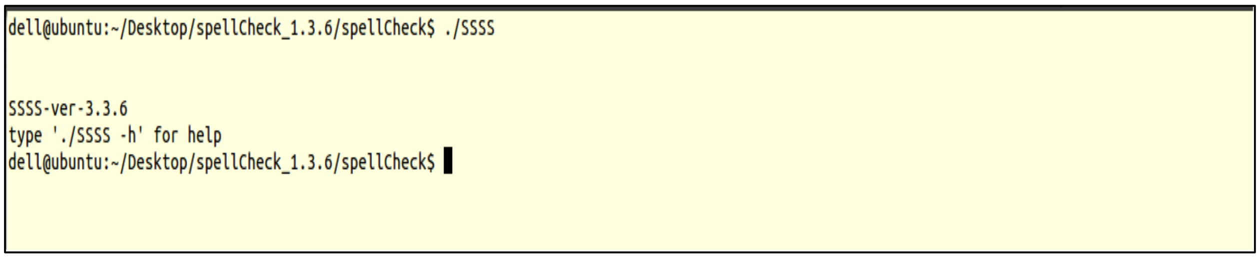


```
SSSS-ver-3.3.6
this program is a simple spell suggester which can suggest a right spelling of incorrectly spelled word. With the help of this program you can ch
eck the entire textfile. You can add word to the database as well as u can print the whole database. one can also get the different possible word
s starting from specific letter.
co-founder: Shashwat Sanghavi
co-founder: Shikhar Pandya

commands
print database -p
search in database -s
insert a word indatabase -i
check a file -f
for help -h
possible words -w
example: ./SSSS -<command> <word/filename>

dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$
```

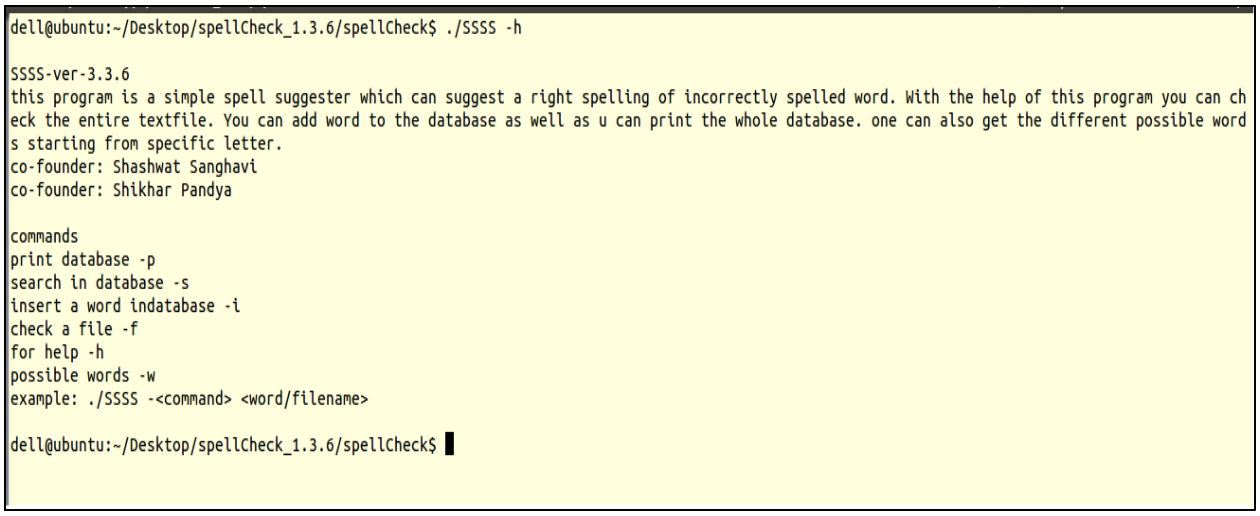
SS2: After the execution of install program, binary file created.



```
dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$ ./SSSS


SSSS-ver-3.3.6
type './SSSS -h' for help
dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$
```

SS3: If command written by the user is wrong or NULL.



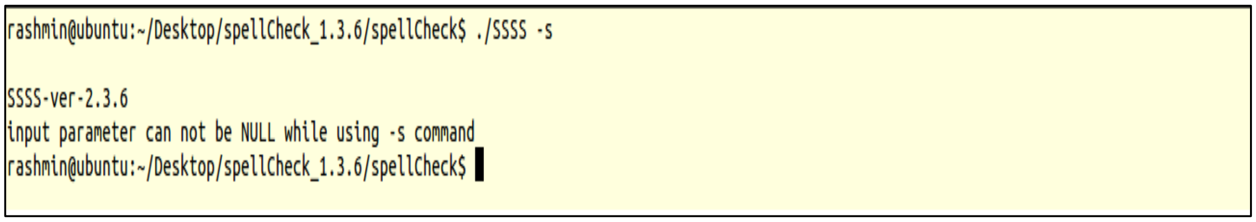
```
dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$ ./SSSS -h

SSSS-ver-3.3.6
this program is a simple spell suggester which can suggest a right spelling of incorrectly spelled word. With the help of this program you can ch
eck the entire textfile. You can add word to the database as well as u can print the whole database. one can also get the different possible word
s starting from specific letter.
co-founder: Shashwat Sanghavi
co-founder: Shikhar Pandya

commands
print database -p
search in database -s
insert a word indatabase -i
check a file -f
for help -h
possible words -w
example: ./SSSS -<command> <word/filename>

dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$
```

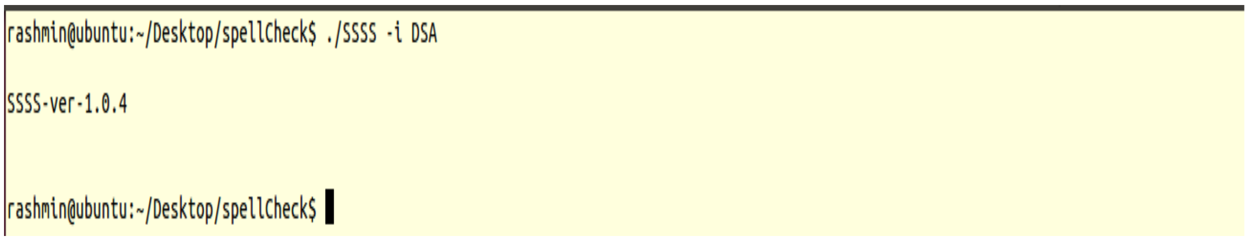
SS4: Help window to provide help for command line interface. Command- ./SSSS –h



```
rashmin@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$ ./SSSS -s

SSSS-ver-2.3.6
input parameter can not be NULL while using -s command
rashmin@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$
```

SS5: For those command line arguments which require a second argument as input and the user does not provide that.

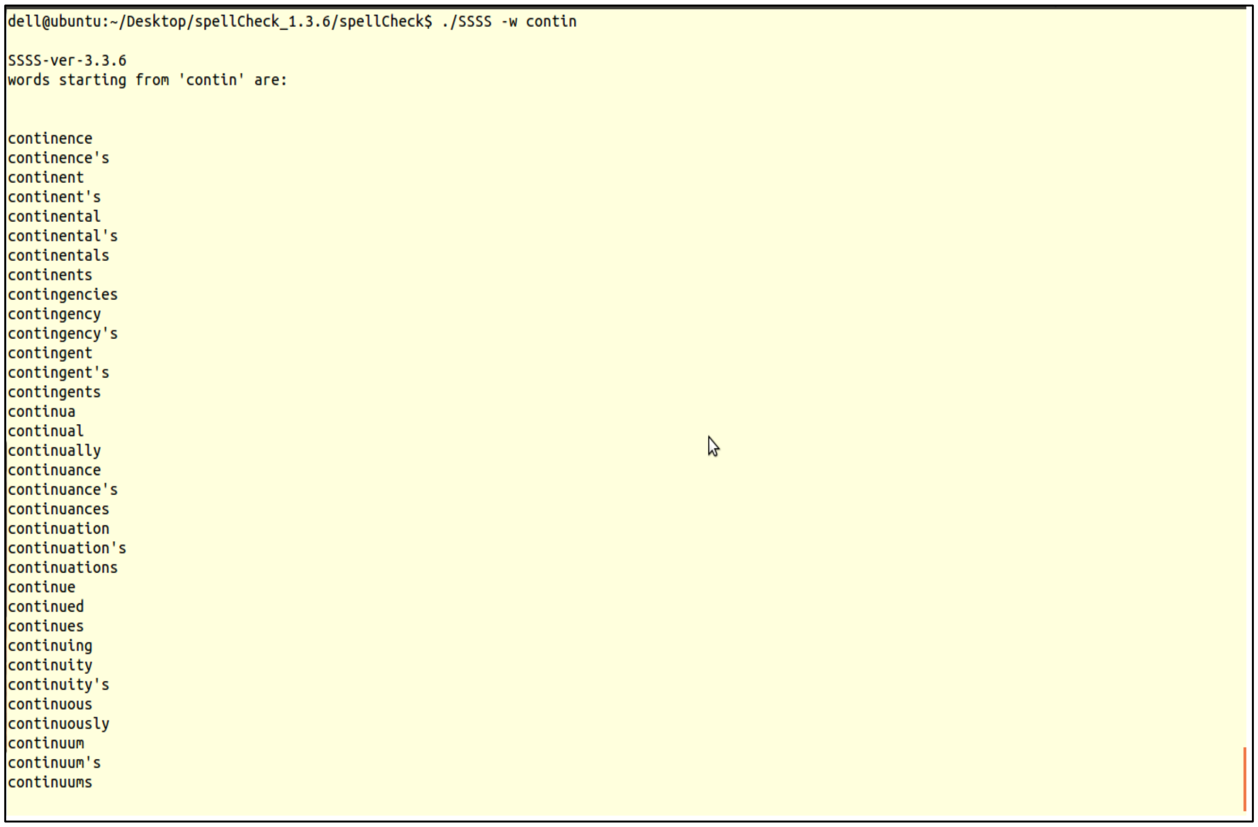


```
rashmin@ubuntu:~/Desktop/spellCheck$ ./SSSS -i DSA

SSSS-ver-1.0.4


rashmin@ubuntu:~/Desktop/spellCheck$
```

SS6: inserting a word DSA into the database. Command ./SSSS –I DSA

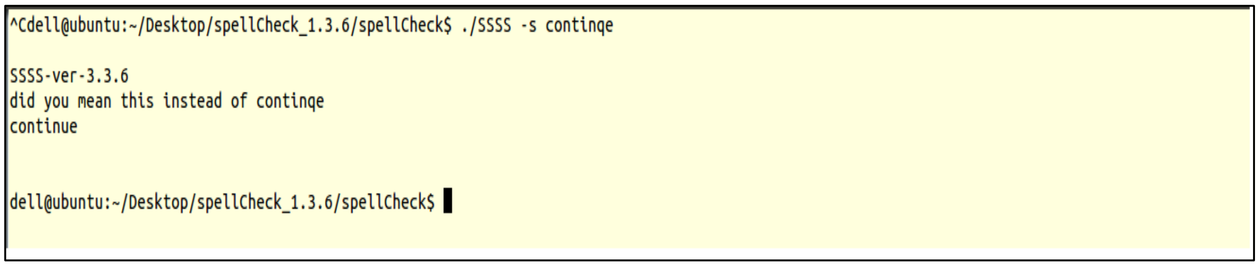


```
dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$ ./SSSS -w contin

SSSS-ver-3.3.6
words starting from 'contin' are:


continence
continence's
continent
continent's
continental
continental's
continentals
continents
contingencies
contingency
contingency's
contingent
contingent's
contingents
continua
continual
continually
continuance
continuance's
continuances
continuation
continuation's
continuations
continue
continued
continues
continuing
continuity
continuity's
continuous
continuously
continuum
continuum's
continuums
```

SS7: AutoComplete functionality. Command-./SSSS –w contin

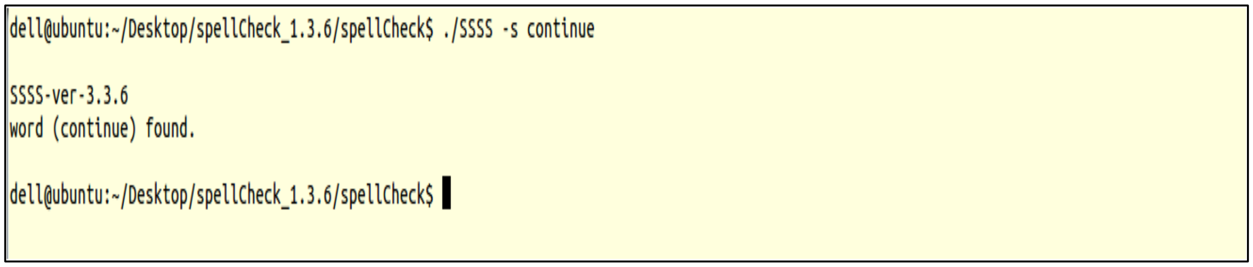


```
^Cdell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$ ./SSSS -s continqe

SSSS-ver-3.3.6
did you mean this instead of continqe
continue


dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$
```

SS8 (a): Searching a given word in database (misspelled word has been input). Command - ./SSSS –s contique

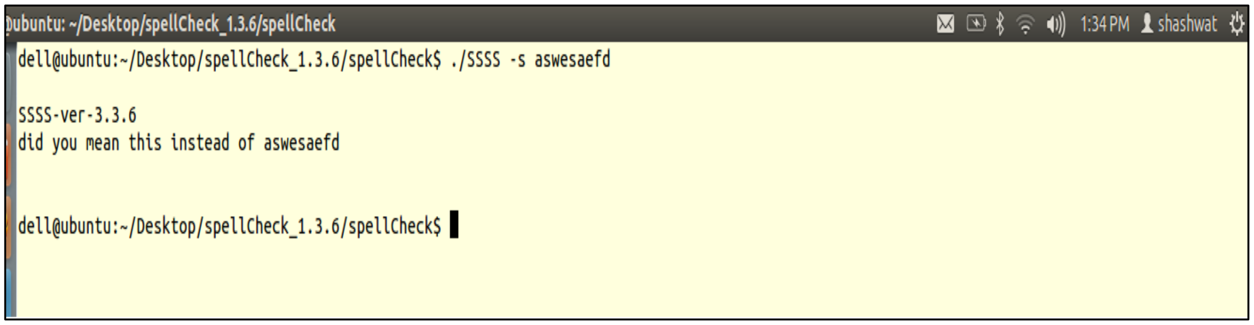


```
dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$ ./SSSS -s continue

SSSS-ver-3.3.6
word (continue) found.

dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$
```

SS8(b): Searching a given word in database (correct word has been input). Command - ./SSSS –s continue

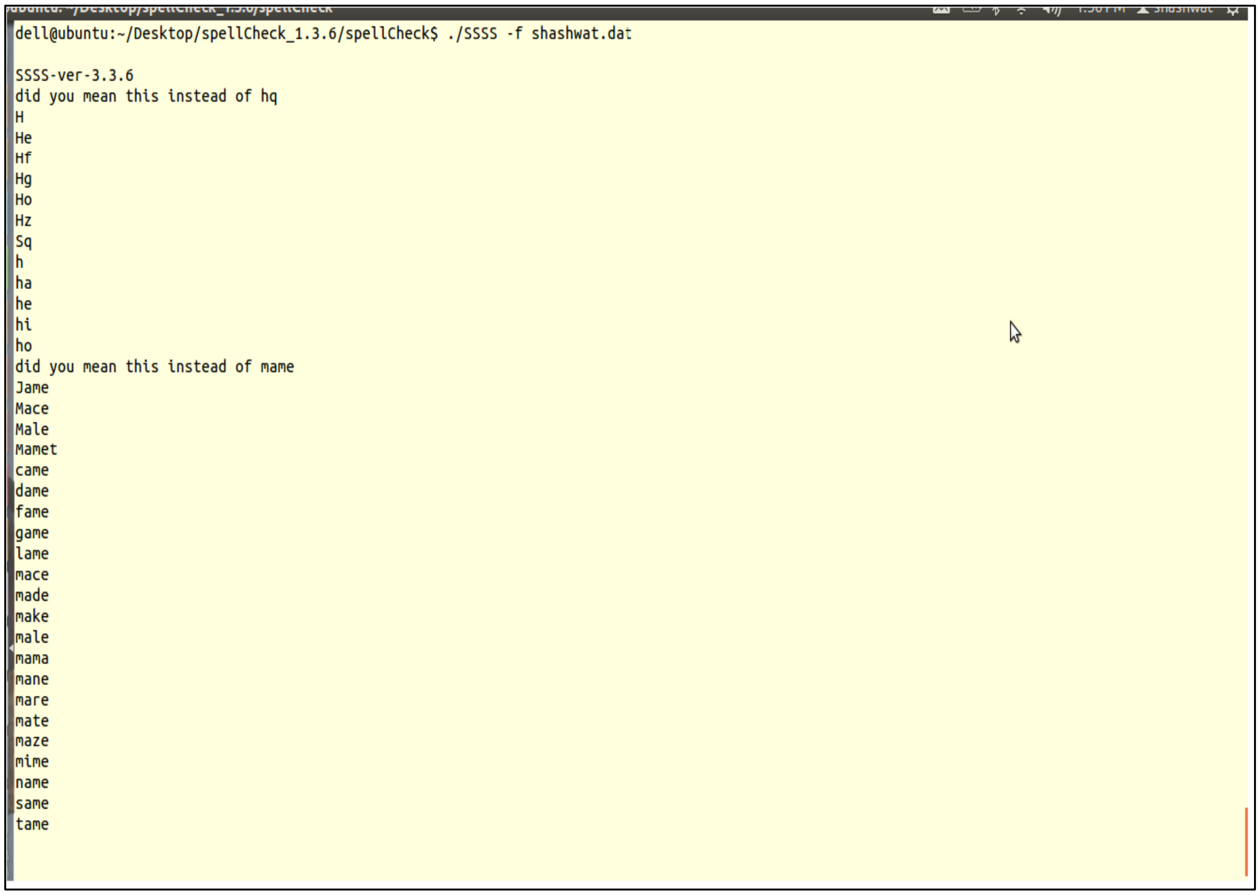


```
dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$ ./SSSS -s aswesaefd

SSSS-ver-3.3.6
did you mean this instead of aswesaefd


dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$
```

SS8(c): Searching a given word in database (arbitary word has been input). Command - ./SSSS –s aswesafd



```
dell@ubuntu:~/Desktop/spellCheck_1.3.6/spellCheck$ ./SSSS -f shashwat.dat

SSSS-ver-3.3.6
did you mean this instead of hq
H
He
Hf
Hg
Ho
Hz
Sq
h
ha
he
hi
ho
did you mean this instead of mame
Jame
Mace
Male
Mamet
came
dame
fame
game
lame
mace
made
make
male
mama
mane
mare
mate
maze
mime
name
same
tame
```

SS9: Checking whole text document named-shashwat.dat. Command-./SSSS -f shashwat.dat

12. References:-

http://staff.science.uva.nl/~andy/ProgC/spell-checker.pdf

http://www.cs.auckland.ac.nz/software/AlgAnim/hash_tables.html

http://stackoverflow.com

http://en.wikipedia.org/wiki/Trie

http://nptel.iitm.ac.in/video.php?subjectId=106102064

http://en.wikipedia.org/wiki/Spell_checker