STRING TOKENIZER

A MINI - PROJECT

Under the guidance of

Prof. Samit Biswas

Department of Computer Science & Technology

Indian Institute of Engineering Science & Technology

Shibpur

Submitted by :-
Nitesh Kumar
(16105108033)
Computer Science & Engineering
Bhagalpur College of Engineering
Bhagalpur

INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR

BONAFIDE CERTIFICATE

This is to certify that this project report entitled “**String Tokenizer**” submitted to **Indian Institute of Engineering Science and Technology, Shibpur** is a bonafide record of work done by “**Nitesh Kumar**” under my supervision from “**1st June, 2018**” to “**30th June, 2018**”.

Samit Biswas
(Assistant Professor)
Department of Computer Science and Technology

Sulata Mitra
(Head of Department)
Department of Computer Science and Technology

Indian Institute of Engineering Science and Technology,
Shibpur, P.O. - Botanic Garden, Howrah – 711103 West Bengal, India.

DECLARATION BY AUTHOR

This is to declare that this report has been written by me. No part of the report is plagiarized from other sources. All information included from other sources have been duly acknowledged. I aver that if any part of the report is found to be plagiarized, I shall take full responsibility for it.

Nitesh Kumar
(Computer Science & Engineering)
(Bhagalpur College of Engineering)

ACKNOWLEDGEMENTS

It is a pleasant duty to express my sincere gratitude to Aryabhatta Knowledge University, Patna for giving me such a wonderful opportunity to complete my first internship project.

I ascribe countless number of thanks to Hon'ble principal, Dr. Nirmal Kumar and TEQIP coordinator Prof. Sanjay Kumar of Bhagalpur College of Engineering, Bhagalpur, for allowing me to join this internship program.

I would like to acknowledge my guide mentor ***Prof. Samit Biswas****, who helped me and guided me throughout the course of completion of this project on "String Tokenizer".*

Lastly, and most importantly, I thank my parents for their love, support and motivation throughout my life.

I also place on record, my sense of gratitude to one and all who, directly or indirectly, have lent their helping hand in this venture.

Nitesh Kumar
(Computer Science & Engineering)
(Bhagalpur College of Engineering)

ABSTRACT

As Google acquired Wavii and Yahoo! acquired Summly, there is no doubt auto summarization technologies have become a hot topic in today's world. So as a computer science freak, one must give hands-on experience on how these technologies actually work.

"String-Tokenizer", as the name itself suggests, this mini-project circles around the analysis of text. The primary motive of this mini-project is to build an even bigger project viz. **"Summarization Technology"**. Today there are two common approaches to "attacking" the summarization mission. The first approach tries to extract the key sentences from the text, and then tries to put them together properly. The second approach, tries to analyze the text, and to rewrite or rephrase it in a short way. So, to proceed in the second approach, one need to work on analysis of text and develop a **"String Tokenizer"** tool.

"String Tokenizer" is nothing but a software or tool which takes input a paragraph from a file (say .txt file) and yields output as an another text file containing the frequency (sorted in descending order of frequency) of each token (words, numbers & symbols) contained in the paragraph. The basics behind the mini-project **"String Tokenizer"** is the analysis of strings.

A great personality said, **"If you want to discover something, keep sharp eyes on the basics"**. Moreover, most of the engineering students start coding from C language, so this project will be using C language itself. There are many predefined string functions provided in C language. However, to know "How exactly the things are working?" the programmer must use their own functions and algorithms. The algorithms used in this project uses only primitive functions of C language.

The project is developed and run in C language by using JetBrain's Integrated Development Environment (IDE) CLion under the C version of C99 (ISO/IEC 9899:1999) on a Windows-10 (version 1803, OS build 17134.122) system.

CONTENTS

CHAPTER: 1 INTRODUCTION

"Before jumping right into the battlefield, one must prepare well for it." So, before going any further, let us have a look on the overview and pre-requisites which may be required while tackling any difficulty during the preparation of **"String Tokenizer".**

The primary motive of this mini-project is to build an even bigger project viz. **"Summarization Technology"**. The summarization technologies is being used readily in the industry by major tech giant like Google, Yahoo etc. To develop such a demanding technology, **"String Tokenizer"** can act as catalyst for the future scope.

The **"String Tokenizer"** starts by taking input from a text file which may require a good understanding of file input-output. This is explained in chapter 3. After this, the character taken input is compared with arrays containing alphabets, numbers and special characters. After all the required comparisons for a particular character, the character is placed at a proper index in an array. For this, the concepts of arrays are used. These are included in chapter 2 and chapter 4. When all the characters from the file are accessed and stored in an array at proper index, the characters from the array are then collected and formed tokens. These tokens are then compared to get the required number of occurrences and the frequencies are stored in another array. For this again, the concepts of arrays are used. For developing proper structure for the program, the idea of finite automata is used. Alongside, some string functions (from header file <string.h>) are used for comparing tokens and some other purposes. This is explained in chapter 5. Then tokens and frequencies are sorted in the descending order of their frequencies. For this purpose, basic knowledge of searching and sorting (data structures) is used. Sorting is explained in chapter 6. After all these operations, the sorted tokens along with their frequencies are then stored in another text file which again requires the concept of file input-output. This is explained in chapter 7.

In short, before moving further, one must have an overview of the concepts of arrays, strings, finite automata, searching and sorting and file input-output.

CHAPTER: 2 ARRAYS

Arrays are a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

"String Tokenizer" consists of three 1-dimensional character arrays each containing set of alphabets, numbers and symbols. Apart from these, there is a 2-dimensional character array which contains all the characters taken input from the text file and a 1-dimensional integer array to store frequency of each token generated during the process. Also, there are four arrays which are used in the code, two for the comparison purpose and other two for the temporary storage of elements of arrays of tokens and frequencies at the time of sorting of elements.

After accessing one character from file, start () function checks whether the input character is an alphabet, a number or a special symbol. Whenever an alphabet is encountered while processing character from the file, the character is passed to the alpha () function which performs the further operations required. Similarly when any number or a special symbol is encountered, the character is passed to the function num (), or sym () respectively as according to the situation. The operations performed by these functions are explained in detail in chapter 4.

SECTION: 2.1 ~ ARRAY OF ALPHABETS

Array of alphabets (named alphabets []) contains all the alphabets of the English language. It contains all the upper case and lower case alphabets. Upper case alphabets have ASCII codes from 65 to 90 and that of lower case have ASCII codes from 97 to 122. So to include all these characters in the array (of alphabets) two loops are used one after the other. The size of this array is 52 indexed from 0 to 51. The C code for this array is given in **APPENDIX- I.i.**

SECTION: 2.2 ~ ARRAY OF NUMBERS

Array of numbers (named numbers []) contains all the numbers from 0 to 9. These numbers have ASCII codes from 48 to 57. So to include all these characters in the array (of numbers) a single loop is used. The size of this array is 10 indexed from 0 to 9. The C code for this array is given in **APPENDIX- I.ii**.

SECTION: 2.3 ~ ARRAY OF SPECIAL SYMBOLS

Array of special characters or symbols (named symbols []) contains all the symbols. It contains all the rest characters having ASCII codes below 256 except that of alphabets (upper case and lower case) and numbers. Since symbols are at different places in the ASCII code table, so to include all these characters in the array (of symbols) various loops are used one after the other. The size of this array is 194 indexed from 0 to 193. The C code for this array is given in **APPENDIX- I.iii**.

SECTION: 2.4 ~ ARRAY OF TOKENS

Array of tokens (named words [] []) contains all the characters taken input from the text file. After the text is analyzed by the comparison functions (start (), alpha () or num () or sym ()), the characters are stored in this array one after the other at the proper index. The comparison and storing processes are explained in chapter 4. The size of this array is defined as maximum row number (defined as 999) and maximum column number (defined as 20). The C code for this array is given in **APPENDIX- III**.

SECTION: 2.5 ~ ARRAY OF FREQUENCIES

Array of frequencies (named freq []), unlike others, is an integer array, contains all the frequencies of the tokens stored in the array of tokens (words [] []). The counting and storing of frequencies processes are explained in chapter 5. The size of this array is defined equal to the maximum row number of the array of tokens, as frequencies of all the tokens are to be stored in this array corresponding to each row. The C code for this array is given in **APPENDIX- IV**.

SECTION: 2.6 ~ TEMPORARY ARRAYS

Apart from all the arrays explained earlier in this chapter, there are two character arrays (strings) which is used to store tokens for a short time. These arrays (string1 [] and string2 []) store tokens for comparison process. Their value keeps on updating during the running state of program.

There are two more arrays, one used for swapping the elements of array of tokens (temp_words [] []) and other for swapping the elements of array frequency (temp_freq []). The size and data types of these arrays are same as the size of the arrays whose data they store for short time and return instantly.

CHAPTER: 3 ACCESSING FILE

Since all the required arrays of alphabets, numbers and symbols are created already, it is the best time to start taking characters input from the text file.

A file represents a sequence of bytes, regardless of it being a text file or a binary file. C programming language provides access on high level functions as well as low level (OS level) calls to handle file on your storage devices. The function **fopen ( )** can be used in read mode (“r”) to open an existing file. This call will initialize an object of the type **FILE**, which contains all the information necessary to control the stream.

SECTION: 3.1 ~ ALGORITHM

3.1.1 FILE NOT FOUND

i. Check if the file is present at the provided path.

ii. If **i.** is found to be false, return error “File not found”.

iii. Stop the execution and exit.

3.1.2 FILE FOUND

i. Take input first character from the text file.

ii. Call the start () function passing the character as argument for further execution.

iii. Repeat steps **i.** and **ii.** until EOF is reached.

iv. Once EOF is reached, Stop taking input and return control to main () function for further execution of the codes.

The codes to access file checks if the file is present at the provided address.

If the provided path does not contain the file, the program returns with an error "File not found!" And if the file is found at the provided path, the file opens successfully and the further execution of the program takes place.

The function keeps on taking input from the file character by character and keeps passing it to the start () function until the EOF (End Of File) is reached. Once the EOF is reached, the function stops taking input any further character and the control of the program proceeds as the main () function directs.

The C code for taking input from the file character by character and passing the character to start () function is given in **APPENDIX- II**.

CHAPTER: 4 COMPARING CHARACTERS

After the characters have started to be taken input from the file successfully, it is the best time to start comparing them and store them at proper index in an array for further execution.

SECTION: 4.1 ~ ALGORITHM

- Check if the input character is an alphabet or a number or a symbol.
- If it is alphabet, pass it as argument to alpha () function.
- If it is number, pass it as argument to num () function.
- If it is symbol, pass it as argument to sym () function.
- Return the value to main () function.

4.1.1 ALPHABETS

i. Check if previous character is also an alphabet.

ii. If **i.** is true, store current character in same row in the array of tokens and return the value to main () function.

iii. If **i.** is false, the value of row number is incremented, column number is set to zero and the character is stored in the array of tokens.

iv. Return the value to main () function.

4.1.2 NUMBERS

i. Check if previous character is also a number.

ii. If **i.** is true, stores current character in same row in the array of tokens and return.

iii. If **i.** is false, the value of row number is incremented, column number is set to zero and the character is stored in the array of tokens.

iv. Return the value to main () function.

4.1.3 SPECIAL SYMBOL

i. The value of row number is incremented, column number is set to zero and the character is stored in the array of tokens.

ii. Returns the value to main () function.

Comparison of characters takes place at two stages in the code. At first whenever a character is taken input from the text file, the comparison of that character is done with the arrays of alphabets, numbers and symbols one by one by start () function. Once it is decided the input character is an alphabet or a number or a special symbol, the character is then passed to their respective functions (alpha () or num () or sym ()) for further execution.

At second stage, if the input character is found to be an alphabet, the previous character in the array of tokens (words []) is checked if it is also an alphabet or not. If the previous alphabet is also an alphabet, then the current character is put in the same row in the array of tokens otherwise the row number is incremented by one and then the current character is stored in the array and control returns to fopen () function for next character input.

If the input character is found to be a number, the previous character in the array of tokens (words []) is checked if it is also a number or not. If the previous alphabet is also a number, then the current character is put in the same row in the array of tokens otherwise the row number is incremented by one and then the current character is stored in the array and control returns to fopen () function for next character input.

If the input character is found to be a symbol, the row number is incremented by one and then the current character is stored in the array and control returns to fopen () function for next character input.

The C codes for comparisons are given in **APPENDIX- III.**

CHAPTER: 5 FREQUENCIES

This chapter is the core chapter of this mini-project. In the previous chapters, the algorithms for accessing the file contents, comparing tokens and storing them in an array at proper position have already been explained. Now it is the time to find how many times a particular token occurred in the file content that is frequency of the tokens.

SECTION: 5.1 ~ ALGORITHM

i. All the elements of the array of frequencies (freq []) is initialized with -1.

ii. Two nested loops are used to access the rows of the array of tokens, to pivot a token at a time and the other to check for the pivoted token in the rest of the content in the array. The index of the inside loop must start from one greater than the pivoted token.

iii. Another loop is initialized inside the second loop, to access all the characters of the selected row of the array of tokens to copy the pivoted token in array string1 [] and other tokens to be compared in the array string2 []. Compare the two arrays (string1 [] and string2 []).

iv. If both the strings are found to be equal, the frequency of second token is set to 0 and the counter is incremented.

v. Now for storing the count of the tokens at the end of comparison, frequency of that element is checked if it is not 0. If it is not 0, then it is stored in the array of frequencies of tokens for all the rows of array of tokens, otherwise it is discarded.

vi. Return the array of frequencies to main () function.

Now when the array of tokens is available, the frequencies of each token can be generated. At first, all the elements of array of frequencies are initialized to -1. Then for comparing tokens, two nested loops are used. First loop act as pivot and stores a particular token at a time and the inner loop check for the duplicate token row by row in the array of tokens.

Inside the outer loop the counter is initialized to 1 (count=1) indicating the particular element has occurred once. Inside the inner loop an another loop is used to traverse through all the row elements of the array of tokens and copy the pivoted token to string1 [] and comparing token to string2 [].

Once the strings are copied, they can be compared using strcmp () function which is included in header file <string.h>. For better precision stricmp () or strcmpi () functions can be used as it ignores the case sensitivity of the alphabets.

Now if the strings are found to be equal, the frequency if inner token is set to 0 and the counter is incremented and the loop is allowed to run for all occurrences of the tokens. Then the counter is stored in the array of frequencies (whenever the frequencies does not have previous value of 0) for all the rows of the array of tokens.

This stores the frequency of each and every token stored in the array of tokens with corresponding index in the array of frequencies. The C code for finding the frequencies of tokens of the array is given in the **APPENDIX- IV**.

Next chapter will explain how the tokens are sorted in a descending order of their frequencies.

CHAPTER: 6 SORTING

Once the frequencies of all the tokens have been found, the next work is to sort the tokens in a particular order, in this case, in descending order of their frequencies.

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order. The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats.

Sorting algorithms may require some extra space for comparison and temporary storage of few data elements. These algorithms do not require any extra space and sorting is said to happen in-place, or for example, within the array itself. This is called **in-place sorting**. Bubble sort is an example of in-place sorting. Bubble sort is used here in this mini-project as well.

SECTION: 6.1 ~ ALGORITHM

i. In Pass 1, first and second elements are compared, then second element is compared with third, and so on. Finally, (N-2) th element is compared with (N-1) th element. Pass 1 involves n–1 comparisons and places the biggest element at the lowest index of the array.

ii. In Pass 2, second element is compared with third, third element is compared with fourth, and so on. Finally, (N–3) th element is compared with (N–2) th element. Pass 2 involves n–2 comparisons and places the second biggest element at the second lowest index of the array.

iii. Same process continues and finally, in Pass n–1, (N-1) th and Nth elements are compared. After this step, all the elements of the array are arranged in descending order.

At first, the frequencies of consecutive tokens are compared one after the other and the frequency with larger value is moved upward in the array of tokens and along with the corresponding frequencies. It is done using the concept of swapping the elements.

Whenever a token with greater frequency is encountered at higher index of the arrays, it is swapped with the uppermost accessible token and corresponding frequency in the loop. This process is repeated until the whole array is traversed and all the tokens and corresponding frequencies are arranged in the required descending order.

This is how the tokens and their frequencies are sorted as according to the required condition. The C code for sorting the tokens according to descending order of frequencies of tokens of the array is given in the **APPENDIX- V**.

In the next chapter, the storing of these sorted arrays of tokens along with their frequencies into a text file is explained in detail.

CHAPTER: 7 STORING TO FILE

Since all the computational work is done, from taking input from a text file to finding frequencies and sorting them accordingly, now it is the right time to store the final outcome to a text file.

To store the tokens and frequencies to a file, the fopen () function can be used in write mode (“w”). In write mode, if the file is not present at the provided path, it is created, and if the file is already present, that file is opened and all the contents, if present, are cleared and then the writing process starts from the starting point of the text file. This call will initialize an object of the type **FILE**, which contains all the information necessary to control the stream.

SECTION: 7.1 ~ ALGORITHM

7.1.1 FILE NOT FOUND

i. Create a file with file name specified in the code.
ii. Start writing the characters as directed by the code.
iii. Returns the control to main () function.

7.1.2 FILE FOUND

i. Open the file.
ii. Clear the data if any present previously in the found file.
iii. Start writing the characters as directed by the code.
iv. Return the control to main () function.

The function to write the tokens and frequencies, first check if the file is already present at the provided address.

If the file is already present, the file is opened and all the previously contained data is erased from the file. If the file is not present at the provided path, then it is created with the same file name as specified in the code.

Then in the opened file, the pointer points to the starting element of the file. Once a character is written in that element, the pointer shifts to the next element of the file. To write the tokens, the function fputc () can be used. On the other hand, for writing the frequencies, an integer type is required to take input, so the function fprintf () function is used with a format specifier for integers (%d). If fputc () is used for inserting the frequencies, the characters corresponding to the integer value is printed instead of the integer itself.

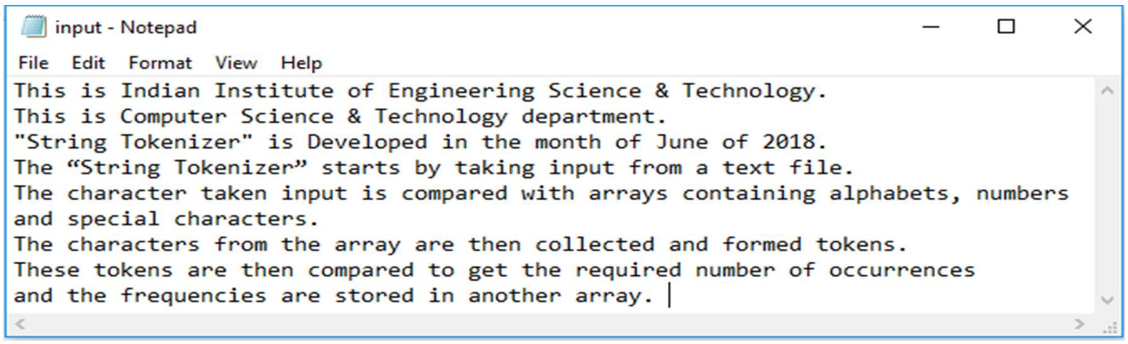
It is to be noted that whichever file is opened in the code, must be closed before the code finishes, failing to which may lead to any unknown ambiguity in the data of the file.

The C code for storing the frequencies of tokens of the array is given in the **APPENDIX- VI**.

CHAPTER: 8 EXPERIMENTAL RESULTS

SECTION: 8.1 ~

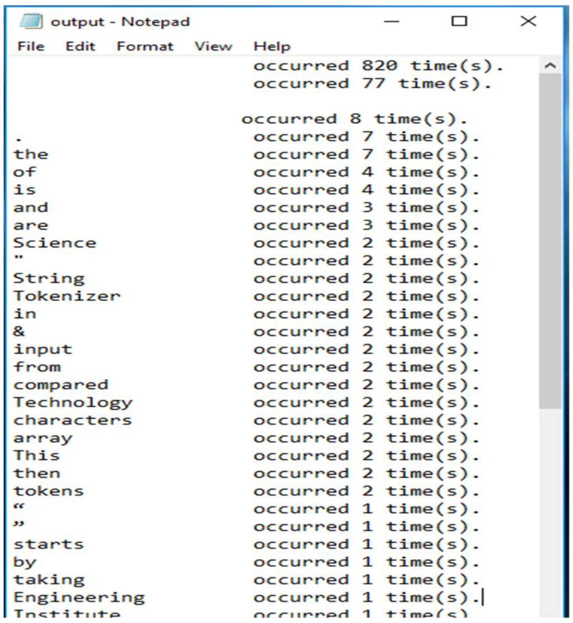
Input file:-



```
This is Indian Institute of Engineering Science & Technology.
This is Computer Science & Technology department.
"String Tokenizer" is Developed in the month of June of 2018.
The “String Tokenizer” starts by taking input from a text file.
The character taken input is compared with arrays containing alphabets, numbers
and special characters.
The characters from the array are then collected and formed tokens.
These tokens are then compared to get the required number of occurrences
and the frequencies are stored in another array.
```

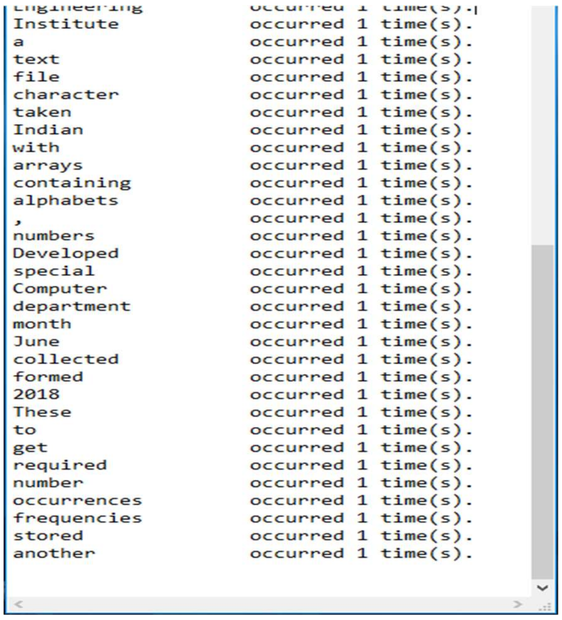
(Figure 8.1.1)

As expected, the “String Tokenizer” produced following output file when the above input file is accessed with it:-



```
                                occurred 820 time(s).
                                occurred 77 time(s).

                               occurred 8 time(s).
.                               occurred 7 time(s).
the                             occurred 7 time(s).
of                              occurred 4 time(s).
is                              occurred 4 time(s).
and                             occurred 3 time(s).
are                             occurred 3 time(s).
Science                         occurred 2 time(s).
"                               occurred 2 time(s).
String                          occurred 2 time(s).
Tokenizer                       occurred 2 time(s).
in                              occurred 2 time(s).
&                               occurred 2 time(s).
input                           occurred 2 time(s).
from                            occurred 2 time(s).
compared                        occurred 2 time(s).
Technology                      occurred 2 time(s).
characters                      occurred 2 time(s).
array                           occurred 2 time(s).
This                            occurred 2 time(s).
then                            occurred 2 time(s).
tokens                          occurred 2 time(s).
“                               occurred 1 time(s).
”                               occurred 1 time(s).
starts                          occurred 1 time(s).
by                              occurred 1 time(s).
taking                          occurred 1 time(s).
Engineering                     occurred 1 time(s).
```



```
Institute                       occurred 1 time(s).
a                               occurred 1 time(s).
text                            occurred 1 time(s).
file                            occurred 1 time(s).
character                       occurred 1 time(s).
taken                           occurred 1 time(s).
Indian                          occurred 1 time(s).
with                            occurred 1 time(s).
arrays                          occurred 1 time(s).
containing                      occurred 1 time(s).
alphabets                       occurred 1 time(s).
,                               occurred 1 time(s).
numbers                         occurred 1 time(s).
Developed                       occurred 1 time(s).
special                         occurred 1 time(s).
Computer                        occurred 1 time(s).
department                      occurred 1 time(s).
month                           occurred 1 time(s).
June                            occurred 1 time(s).
collected                       occurred 1 time(s).
formed                          occurred 1 time(s).
2018                            occurred 1 time(s).
These                           occurred 1 time(s).
to                              occurred 1 time(s).
get                             occurred 1 time(s).
required                        occurred 1 time(s).
number                          occurred 1 time(s).
occurrences                     occurred 1 time(s).
frequencies                     occurred 1 time(s).
stored                          occurred 1 time(s).
another                         occurred 1 time(s).
```

(Figure 8.1.2) (Figure 8.1.3)

In the output file above, “occurred 820 time(s)” in the first line, shows the count of blank elements of the array of tokens, “occurred 77 time(s)” in the second line, shows the count of spaces (ASCII code 32) and “occurred 8 time(s)” in the third line, shows the count of new lines (“\n”) in the array of tokens. Rest characters can clearly be seen.

The C code for whole program is given in the **APPENDIX- VII**.

CHAPTER: 9 CONCLUSIONS AND FUTURE SCOPE

SECTION: 9.1 ~

As the results of the test run of the code clearly depicts, the **“String Tokenizer”** stands up totally on the expectations and does the work as explained earlier. This shows the successful completion of the project with positive outcomes. This mini-project can now be extended further for the development of **“Summarization Technology”**.

SECTION: 9.2 ~

The future of the **“String Tokenizer”** is the **“Summarization Technology”**. The **“String Tokenizer”** can be used to develop an even larger project which is used for summarizing the larger paragraph into a smaller one. The summarization technologies is being used readily in the industry by major tech giant like Google, Yahoo etc. The **“Summarization Technology”** has a vast future in the world of Artificial Intelligence. To develop such a demanding technology, **“String Tokenizer”** can act as catalyst for the future scope.

CHAPTER: 10 REFERENCES AND BIBLIOGRAPHY

1. **The C Programming Language - 2nd Edition – Brian W. Kernighan and Dennis M. Ritchie.**
2. **Introduction to Automata Theory, Languages, and Computations (2006, Prentice Hall) - John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman**
3. **https://www.iist.ac.in/sites/default/files/projectinternship/Internship%20Report%20Format_1.pdf**
4. **https://www.tutorialspoint.com/cprogramming/c_arrays.htm**
5. **https://www.youtube.com/watch?v=csEv5ZRMvpQ**
6. **http://www.techcrashcourse.com/2015/11/c-program-to-find-frequency-each-array-elements.html**
7. **https://www.tutorialspoint.com/data_structures_algorithms/sorting_algorithms.htm**

APPENDICES

APPENDIX- I.i

//Array of alphabets

```
 i=0;
 for (c=65;c<=90;c++)
{
alphabets[i]=((char)c);
i++;
}
 for (c=97;c<=122;c++)
{
alphabets[i]=((char)c);
i++;
}
```

APPENDIX- I.ii

//Array of numbers

```
 i=0;
for (c=48;c<=57;c++)
 {
   numbers[i]=((char)c);
 i++;
   }
```

APPENDIX- I.iii

```
//Array of symbols
          i=0;
         for (c=0;c<=47;c++)
          {
           symbols[i]=((char)c);
             i++;
           }
           for (c=58;c<=64;c++)
           {
              symbols[i]=((char)c);
              i++;
           }
           for (c=91;c<=96;c++)
           {
              symbols[i]=((char)c);
              i++;
           }
           for (c=123;c<=256;c++)
           {
              symbols[i]=((char)c);
              i++;
           }
```

APPENDIX- II

//Accessing file (.txt)

```
        fp=fopen("input.txt","r");
```

//File is not present at the provided address

```
         if(fp==NULL)
      {
       printf("\nFile Not Found!\n");
       return 1;
      }
```

//File present at the provided address

```
         else
      {
         while ((string=(char)fgetc(fp))!=EOF)
         {
            start(string);
            string_length++;                   //Counting total number of characters
         }
      }
```

APPENDIX- III

```
//Testing Character taken input
        int start(char str)
        {
        for(i=0;i<=51;i++)
            if(str==alphabets[i])
            {
               alpha(str);
               return words[j][k];
            }
          for(i=0;i<=10;i++)
              if(str==numbers[i])
                 {
                 num(str);
                 return words[j][k];
                 }
        for(i=0;i<=194;i++)
                 if(str==symbols[i])
                 {
                  sym(str);
                 return words[j][k];
                 }
        return words[j][k];
        }
```

//Testing Alphabets

```
int alpha(char stra)
{
   for(i=0;i<=51;i++)
   {
      if (words[j][k] == alphabets[i])
      {
         k++;
         words[j][k] = stra;
         return words[j][k];
      }
   }
   j++;
   k=0;
   words[j][k] = stra;
   return words[j][k];
}
```

//Testing Symbols

```
int sym(char strs)
{
   j++;
   k=0;
   words[j][k]=strs;
   return words[j][k];
}
```

//Testing Numbers

```
int num(char strn)
{
   for(i=0;i<=10;i++)
   {
      if(words[j][k]==numbers[i])
      {
         k++;
         words[j][k]=strn;
         return words[j][k];
      }
   }
   j++;
   k=0;
   words[j][k] = strn;
   return words[j][k];
}
```

APPENDIX- IV

```
//Finding frequency of each word/number/character

            int frequency()

            {

               int count=0,l=0;

               char string1[COL_SIZE],string2[COL_SIZE];

//Initialize each element of frequency array

               for(j=0;j<=ROW_SIZE;j++)

               {

                  freq[j]=-1;

               }

//Counting frequency of words/numbers/symbols

               for(j=0;j<=ROW_SIZE;j++)

               {

                  count=1;

                  for(l=j+1;l<=ROW_SIZE;l++)

                  {

                     k=0;

                     while(k<COL_SIZE)

                     {
```

```
            string1[k]=words[j][k];             //Storing pivot word to string1
            string2[k]=words[l][k];             //Storing comparing word to string2
            k++;
         }
         if(stricmp(string1,string2)==0)
         {
            freq[l]=0;
            count++;
         }
      }
      if(freq[j]!=0)
      {
         freq[j]=count;
      }
   }
   return freq[j];
}
```

APPENDIX- V

```
//Swapping words to show them in descending order of frequency

        for(i=0;i<=ROW_SIZE;i++)
        {
          for(j=i+1;j<=ROW_SIZE;j++)
          {
            if(freq[i]<freq[j])
            {
              for(k=0;k<COL_SIZE;k++)          //Swapping Words
              {
                temp_words[i][k]=words[i][k];
                words[i][k]=words[j][k];
                words[j][k]=temp_words[i][k];
              }
              temp_freq[i]=freq[i];          //Swapping Frequency
              freq[i]=freq[j];
              freq[j]=temp_freq[i];
            }
          }
        }
```

APPENDIX- VI

//Storing words and frequency to a file

```
fp1=fopen("output.txt","w");
for(j=0;j<=ROW_SIZE;j++)
{
   if(freq[j]!=0)
   {
      for(k=0;k<COL_SIZE;k++)
      {
         fputc(words[j][k],fp1);              //Words
      }
      fprintf(fp1,"occurred"),
      fprintf(fp1," %d ",freq[j]);            //Frequency
      fprintf(fp1,"time(s).\n");
   }
}
```

APPENDIX- VII

```
#include <stdio.h>
#include <string.h>
#define ROW_SIZE 999
#define COL_SIZE 20

//Global variables
int numbers[10],alphabets[51],symbols[194],freq[ROW_SIZE];
int i=0,j=0,k=0;
char words[ROW_SIZE][COL_SIZE];

//Testing Alphabets
int alpha(char stra)
{
  for(i=0;i<=51;i++)
  {
    if (words[j][k] == alphabets[i])
    {
      k++;
      words[j][k] = stra;
      return words[j][k];
    }
  }
  j++;
  k=0;
  words[j][k] = stra;

  return words[j][k];
}
```

//Testing Numbers

```
int num(char strn)
{
   for(i=0;i<=10;i++)
   {
      if(words[j][k]==numbers[i])
      {
         k++;
         words[j][k]=strn;
         return words[j][k];
      }
   }
   j++;
   k=0;
   words[j][k] = strn;

   return words[j][k];
}
```

//Testing Symbols

```
int sym(char strs)
{
   j++;
   k=0;
   words[j][k]=strs;
   return words[j][k];
}
```

//Testing Character taken input

```
int start(char str)
{
   for(i=0;i<=51;i++)
      if(str==alphabets[i])
      {
         alpha(str);
         return words[j][k];
      }
   for(i=0;i<=10;i++)
      if(str==numbers[i])
      {
         num(str);
         return words[j][k];
      }
   for(i=0;i<=194;i++)
      if(str==symbols[i])
      {
         sym(str);
         return words[j][k];
      }
   return words[j][k];
}
```

//Finding frequency of each word/number/character

```
int frequency()
{
   int count=0,l=0;
   char string1[COL_SIZE],string2[COL_SIZE];
```

//Initialize each element of frequency array

```
   for(j=0;j<=ROW_SIZE;j++)
   {
      freq[j]=-1;
   }
```

//Counting frequency of words/numbers/symbols

```
   for(j=0;j<=ROW_SIZE;j++)
   {
      count=1;
      for(l=j+1;l<=ROW_SIZE;l++)
      {
         k=0;
         while(k<COL_SIZE)
         {
            string1[k]=words[j][k];            //Storing pivot word to string1
            string2[k]=words[l][k];            //Storing comparing word to string2
            k++;
         }
         if(stricmp(string1,string2)==0)
         {
            freq[l]=0;
            count++;
         }
      }
      if(freq[j]!=0)
      {
         freq[j]=count;
      }
   }

   return freq[j];
}
```

```
        //<<=================================================>>
        //<<=================================================>>
```

//Main() function

```
        int main()
        {
          int string_length=0,c=0,temp_freq[ROW_SIZE];
          char string,temp_words[ROW_SIZE][COL_SIZE];

          FILE *fp,*fp1;                          //Pointers to file
```

//Creating arrays of alphabets/numbers/symbols

//Array of alphabets

```
          i=0;
          for (c=65;c<=90;c++)
          {
            alphabets[i]=((char)c);
            i++;
          }
          for (c=97;c<=122;c++)
          {
            alphabets[i]=((char)c);
            i++;
          }
```

```
//Array of numbers
            i=0;
            for (c=48;c<=57;c++)
            {
               numbers[i]=((char)c);
               i++;
            }

//Array of symbols
            i=0;
            for (c=0;c<=47;c++)
            {
               symbols[i]=((char)c);
               i++;
            }
            for (c=58;c<=64;c++)
            {
               symbols[i]=((char)c);
               i++;
            }
            for (c=91;c<=96;c++)
            {
               symbols[i]=((char)c);
               i++;
            }
            for (c=123;c<=256;c++)
            {
               symbols[i]=((char)c);
               i++;
            }
```

//Accessing file (.txt)

```
        fp=fopen("input.txt","r");
```

//If File is not present at the provided address

```
        if(fp==NULL)
        {
          printf("\nFile Not Found!\n");
          return 1;
        }
```

//File present at the provided address

```
        else
        {
          while ((string=(char)fgetc(fp))!=EOF)
          {
            start(string);
            string_length++;            //Counting total number of characters
          }
        }
        printf("\nFile contains %d characters.\n\n",string_length);
```

//Calling frequency function

```
        frequency();
```

//Swapping words to show them in descending order of frequency

```
        for(i=0;i<=ROW_SIZE;i++)
        {
          for(j=i+1;j<=ROW_SIZE;j++)
          {
            if(freq[i]<freq[j])
```

```
                {
                    for(k=0;k<COL_SIZE;k++)              //Swapping Words
                    {
                        temp_words[i][k]=words[i][k];
                        words[i][k]=words[j][k];
                        words[j][k]=temp_words[i][k];
                    }
                    temp_freq[i]=freq[i];              //Swapping Frequency
                    freq[i]=freq[j];
                    freq[j]=temp_freq[i];
                }
            }
        }
```

//Storing words and frequency to a file

```
        fp1=fopen("output.txt","w");
        for(j=0;j<=ROW_SIZE;j++)
        {
            if(freq[j]!=0)
            {
                for(k=0;k<COL_SIZE;k++)
                {
                    fputc(words[j][k],fp1);              //Words
                }
                fprintf(fp1,"occurred"),
                fprintf(fp1," %d ",freq[j]);              //Frequency
                fprintf(fp1,"time(s).\n");
            }
        }
```

```
//Printing words with frequency to std out
            for(j=0;j<=ROW_SIZE;j++)
            {
               if(freq[j]!=0)
               {
                  for(k=0;k<COL_SIZE;k++)
                  {
                     printf("%c",words[j][k]);      //Words
                  }
                  printf("occurred %d time(s). \n",freq[j]);      //Frequency
               }
            }

//Close open files
            fclose(fp);
            fclose(fp1);

            return 0;
         }


//<<==============================================================>>
//<<============================THE-END=============================>>
```