RAMAIAH UNIVERSITY
OF APPLIED SCIENCES

ASSIGNMENT

Course Code	CSC302A
Course Name	Operating system
Programme	B.tech
Department	CSE
Faculty	FET

Name of the Student	Shikhar singh
Reg. No	17ETCS002168
Semester/Year	05/2017
Course Leader/s	Ms. Naveeta Rani

Declaration Sheet

Student Name	Shikhar singh		
Reg. No	17ETCS002168		
Programme	B.Tech	Semester/Year	05th /2017
Course Code	CSC302A		
Course Title	Operating system		
Course Date		to	
Course Leader	Ms. Naveeta Rani		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

< The table numbers have to be based on the chapter number>

< The Figure numbers have to be based on the chapter number>

Symbol	Description	Units
A	Current	Amp
g	Acceleration due to gravity - 9.81	m/s^2
V	Voltage	Volts
w	Width	mm

< Arrange in alphabetical order>

Question No. 1

Solution to Question No. 1:

1.1 Development of application using sequential approach:

A sequential approach leads to step by step program execution i.e. each functionality is executed after the preceding functionality. It is a simple programming approach, easier to execute but it increases the time complexity of the program as one process has to wait until the previous one gets over.

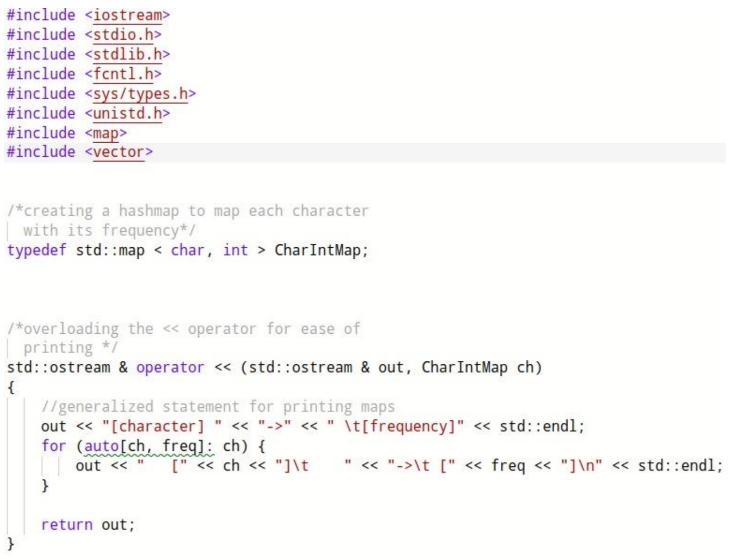
Algorithm for charcount():

```
1. Start
2. Open file passed through function argument and store the return value in an integer variable.
3. Check if the integer variable is greater than 0
      a. If greater than zero, print error in opening file
      b. Else, continue
4. Create a buffer
5. read the opened file into buffer byte by byte until the end of file.
      a. If end of file is reached, break out of loop
      b. Else, map the character key and update the corresponding value.
6. End
```

Algorithm for driver function (main):

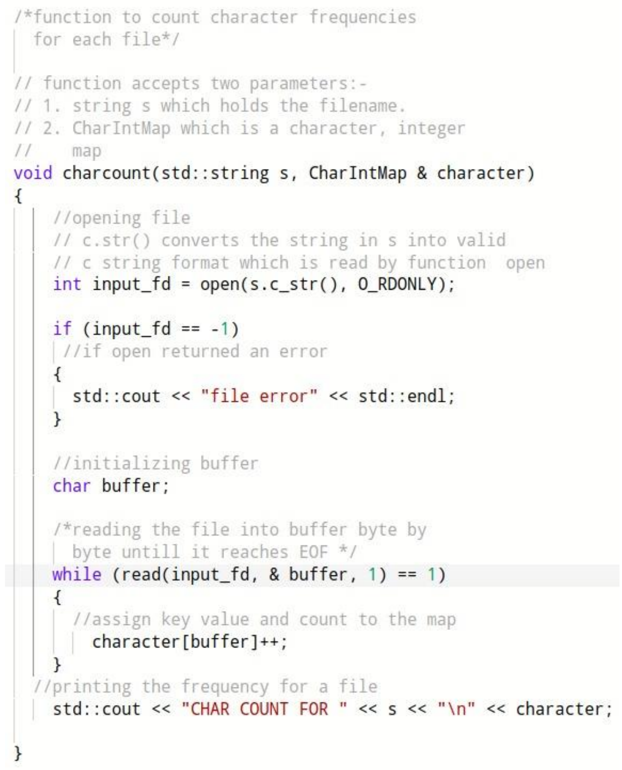
```
1. Create an array of character integer map
2. Print the introductory note
3. Call the charcount() function with suitable arguments i.e.
   1. the file name in c string format and
   2. The character map with proper array index.
4. create another character integer map to calculate total frequency
5. run a loop for each character in function charcount()
      a. run another loop for each [key, value] pair in charcount()
      b. check if key for each character in the file exists.
            i. if it exists, update the value corresponding to it
            ii. else, add the [key, value] to the total map.
7. Print out the individual frequencies for each file and then the total frequency of all the files.
```

SOURCE CODE:



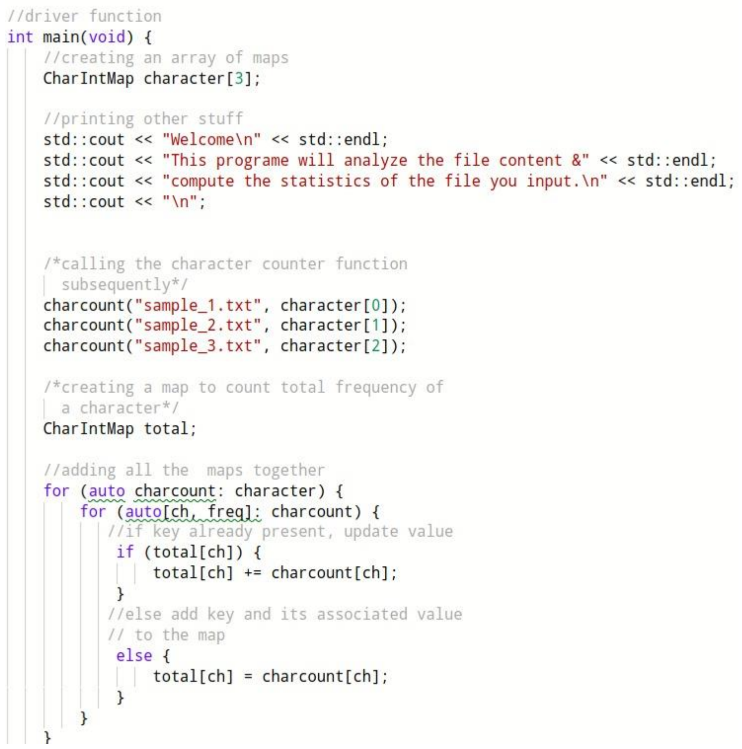
```cpp
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include <map>
#include <vector>


/*creating a hashmap to map each character
 with its frequency*/
typedef std::map < char, int > CharIntMap;



/*overloading the << operator for ease of
 printing */
std::ostream & operator << (std::ostream & out, CharIntMap ch)
{
    //generalized statement for printing maps
    out << "[character] " << "->" << " \t[frequency]" << std::endl;
    for (auto[ch, freq]: ch) {
        out << "    [" << ch << "]\t    " << "->\t [" << freq << "]\n" << std::endl;
    }

    return out;
}
```

Figure 1: source code for character integer map and overloaded << operator



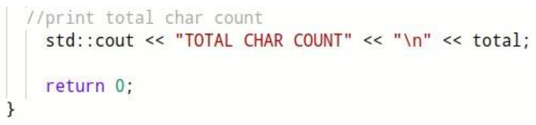
```cpp
/*function to count character frequencies
 for each file*/

// function accepts two parameters:-
// 1. string s which holds the filename.
// 2. CharIntMap which is a character, integer
//    map
void charcount(std::string s, CharIntMap & character)
{
    //opening file
    // c.str() converts the string in s into valid
    // c string format which is read by function  open
    int input_fd = open(s.c_str(), O_RDONLY);

    if (input_fd == -1)
     //if open returned an error
    {
      std::cout << "file error" << std::endl;
    }

    //initializing buffer
    char buffer;

    /*reading the file into buffer byte by
      byte untill it reaches EOF */
    while (read(input_fd, & buffer, 1) == 1)
    {
      //assign key value and count to the map
        character[buffer]++;
    }
  //printing the frequency for a file
    std::cout << "CHAR COUNT FOR " << s << "\n" << character;

}
```

Figure 2: source code for charcount( ) function



```
//driver function
int main(void) {
    //creating an array of maps
    CharIntMap character[3];

    //printing other stuff
    std::cout << "Welcome\n" << std::endl;
    std::cout << "This programe will analyze the file content &" << std::endl;
    std::cout << "compute the statistics of the file you input.\n" << std::endl;
    std::cout << "\n";


    /*calling the character counter function
     subsequently*/
    charcount("sample_1.txt", character[0]);
    charcount("sample_2.txt", character[1]);
    charcount("sample_3.txt", character[2]);

    /*creating a map to count total frequency of
     a character*/
    CharIntMap total;

    //adding all the  maps together
    for (auto charcount: character) {
        for (auto[ch, freq]: charcount) {
            //if key already present, update value
            if (total[ch]) {
                total[ch] += charcount[ch];
            }
            //else add key and its associated value
            // to the map
            else {
                total[ch] = charcount[ch];
            }
        }
    }
```

Figure 3.a.: source code for main driver function

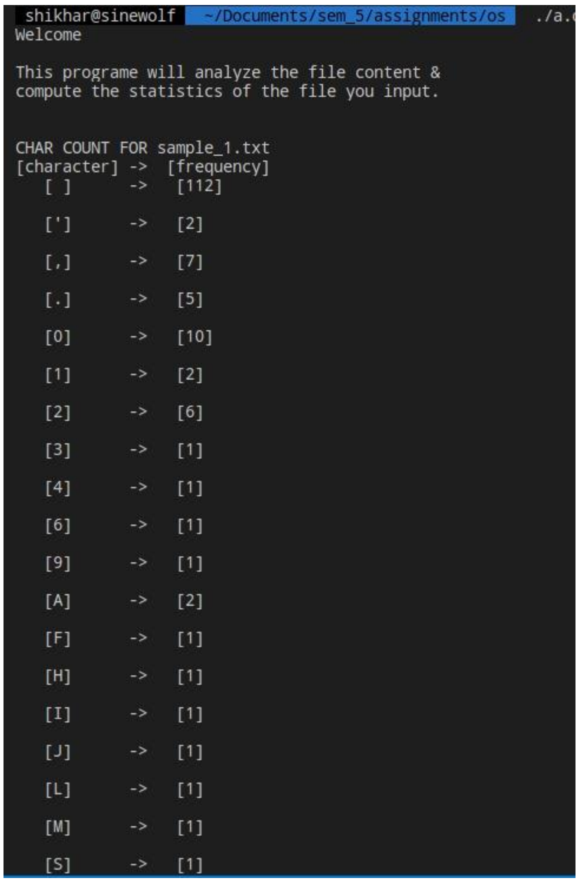


```
    //print total char count
    std::cout << "TOTAL CHAR COUNT" << "\n" << total;

    return 0;
}
```

Figure 3.b: source code for main driver function

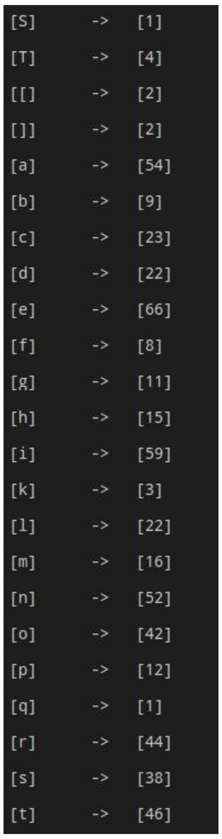
In order to run this source code, we took 3 sample files which are shown in the appendix.
This program reads every character in the sample file and prints the frequency of every character in this file. Also, the main function calculates the total frequency of every character in all the 3 files.

OUTPUT SCREENSHOTS:



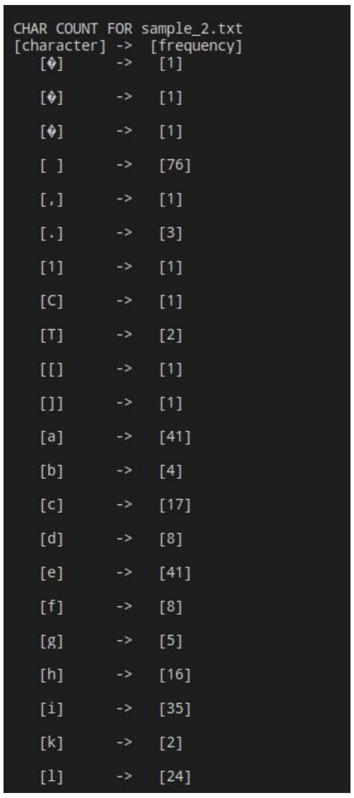
```
 shikhar@sinewolf   ~/Documents/sem_5/assignments/os   ./a.
Welcome

This programe will analyze the file content &
compute the statistics of the file you input.


CHAR COUNT FOR sample_1.txt
[character] ->  [frequency]
   [ ]      ->   [112]

   [']      ->   [2]

   [,]      ->   [7]

   [.]      ->   [5]

   [0]      ->   [10]

   [1]      ->   [2]

   [2]      ->   [6]

   [3]      ->   [1]

   [4]      ->   [1]

   [6]      ->   [1]

   [9]      ->   [1]

   [A]      ->   [2]

   [F]      ->   [1]

   [H]      ->   [1]

   [I]      ->   [1]

   [J]      ->   [1]

   [L]      ->   [1]

   [M]      ->   [1]

   [S]      ->   [1]
```

Figure 4: output



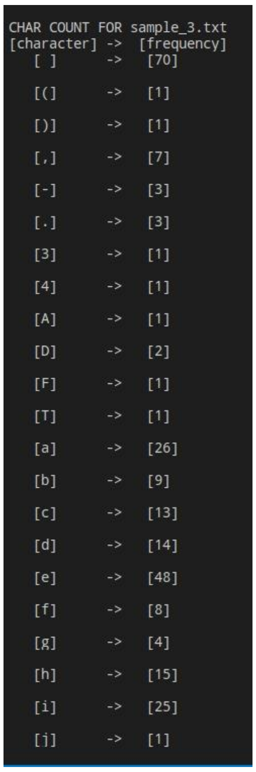
```
[S]       ->   [1]
[T]       ->   [4]
[[]       ->   [2]
[]]       ->   [2]
[a]       ->   [54]
[b]       ->   [9]
[c]       ->   [23]
[d]       ->   [22]
[e]       ->   [66]
[f]       ->   [8]
[g]       ->   [11]
[h]       ->   [15]
[i]       ->   [59]
[k]       ->   [3]
[l]       ->   [22]
[m]       ->   [16]
[n]       ->   [52]
[o]       ->   [42]
[p]       ->   [12]
[q]       ->   [1]
[r]       ->   [44]
[s]       ->   [38]
[t]       ->   [46]
```

Figure 5:output



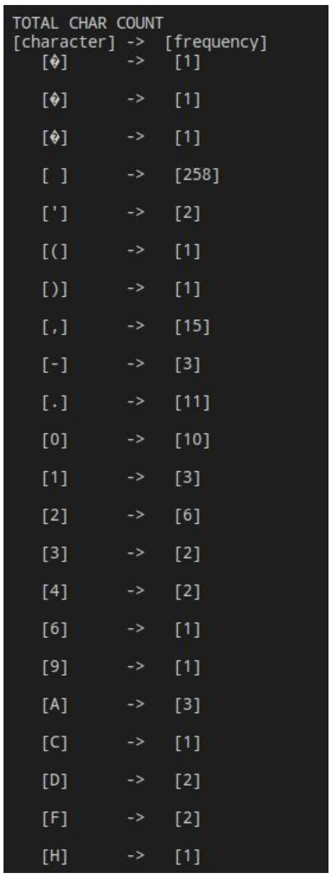
```
CHAR COUNT FOR sample_2.txt
[character] ->  [frequency]
   [�]       ->   [1]
   [�]       ->   [1]
   [�]       ->   [1]
   [ ]       ->   [76]
   [,]       ->   [1]
   [.]       ->   [3]
   [1]       ->   [1]
   [C]       ->   [1]
   [T]       ->   [2]
   [[]       ->   [1]
   []]       ->   [1]
   [a]       ->   [41]
   [b]       ->   [4]
   [c]       ->   [17]
   [d]       ->   [8]
   [e]       ->   [41]
   [f]       ->   [8]
   [g]       ->   [5]
   [h]       ->   [16]
   [i]       ->   [35]
   [k]       ->   [2]
   [l]       ->   [24]
```

Figure 6:output



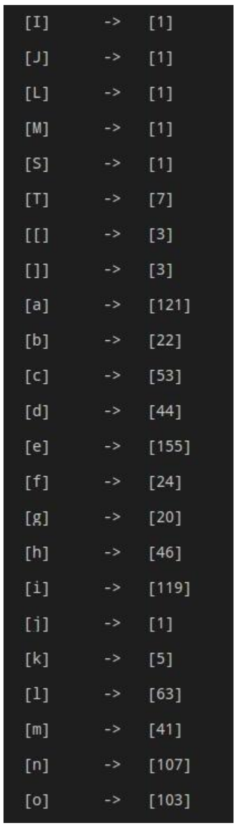
```
CHAR COUNT FOR sample_3.txt
[character] ->  [frequency]
   [ ]      ->   [70]

   [(]      ->   [1]

   [)]      ->   [1]

   [,]      ->   [7]

   [-]      ->   [3]

   [.]      ->   [3]

   [3]      ->   [1]

   [4]      ->   [1]

   [A]      ->   [1]

   [D]      ->   [2]

   [F]      ->   [1]

   [T]      ->   [1]

   [a]      ->   [26]

   [b]      ->   [9]

   [c]      ->   [13]

   [d]      ->   [14]

   [e]      ->   [48]

   [f]      ->   [8]

   [g]      ->   [4]

   [h]      ->   [15]

   [i]      ->   [25]

   [j]      ->   [1]
```

Figure 7:output



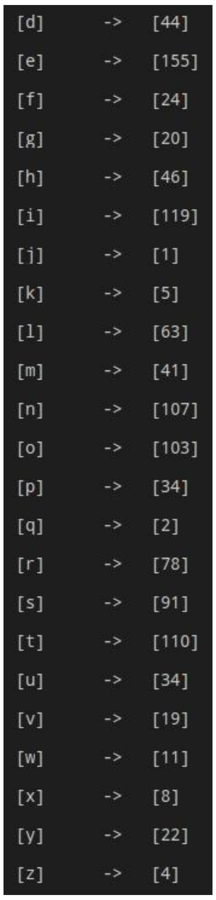
```
TOTAL CHAR COUNT
[character] ->  [frequency]
   [�]      ->   [1]

   [�]      ->   [1]

   [�]      ->   [1]

   [ ]      ->   [258]

   [']      ->   [2]

   [(]      ->   [1]

   [)]      ->   [1]

   [,]      ->   [15]

   [-]      ->   [3]

   [.]      ->   [11]

   [0]      ->   [10]

   [1]      ->   [3]

   [2]      ->   [6]

   [3]      ->   [2]

   [4]      ->   [2]

   [6]      ->   [1]

   [9]      ->   [1]

   [A]      ->   [3]

   [C]      ->   [1]

   [D]      ->   [2]

   [F]      ->   [2]

   [H]      ->   [1]
```

Figure 8:output



```
[I]      ->  [1]
[J]      ->  [1]
[L]      ->  [1]
[M]      ->  [1]
[S]      ->  [1]
[T]      ->  [7]
[[]      ->  [3]
[]]      ->  [3]
[a]      ->  [121]
[b]      ->  [22]
[c]      ->  [53]
[d]      ->  [44]
[e]      ->  [155]
[f]      ->  [24]
[g]      ->  [20]
[h]      ->  [46]
[i]      ->  [119]
[j]      ->  [1]
[k]      ->  [5]
[l]      ->  [63]
[m]      ->  [41]
[n]      ->  [107]
[o]      ->  [103]
```

Figure 9:output



```
[d]      ->  [44]
[e]      ->  [155]
[f]      ->  [24]
[g]      ->  [20]
[h]      ->  [46]
[i]      ->  [119]
[j]      ->  [1]
[k]      ->  [5]
[l]      ->  [63]
[m]      ->  [41]
[n]      ->  [107]
[o]      ->  [103]
[p]      ->  [34]
[q]      ->  [2]
[r]      ->  [78]
[s]      ->  [91]
[t]      ->  [110]
[u]      ->  [34]
[v]      ->  [19]
[w]      ->  [11]
[x]      ->  [8]
[y]      ->  [22]
[z]      ->  [4]
```

Figure 10:output

The figures above shows the character and their corresponding frequencies in each file and all the three files in the following format:

[character] -> [frequency]

1.2 Development of application using multithreaded approach:

In a multithreaded approach, all the functionalities are executed parallelly using different threads for each functionality. The thread copies the entire function and executes it as a process. Multiple threads are able to do the same thing parallelly. Using multithreaded approach is complex and difficult to implement. But it reduces the total execution time drastically since all the functionalities of the program are executed parallelly.

Here, we are using structure to make a bundle for the arguments of the charcount function since the pthread create function will only accept one function argument as its argument.

Algorithm for structure package:

```
1. Declare the string variable for file name.
2. Create an object for character integer map.
3. Declare an arrayed object for the structure.
```

The function is a pointer function called through pthread create which accepts pointer arguments of void datatype.

Algorithm for charcount():

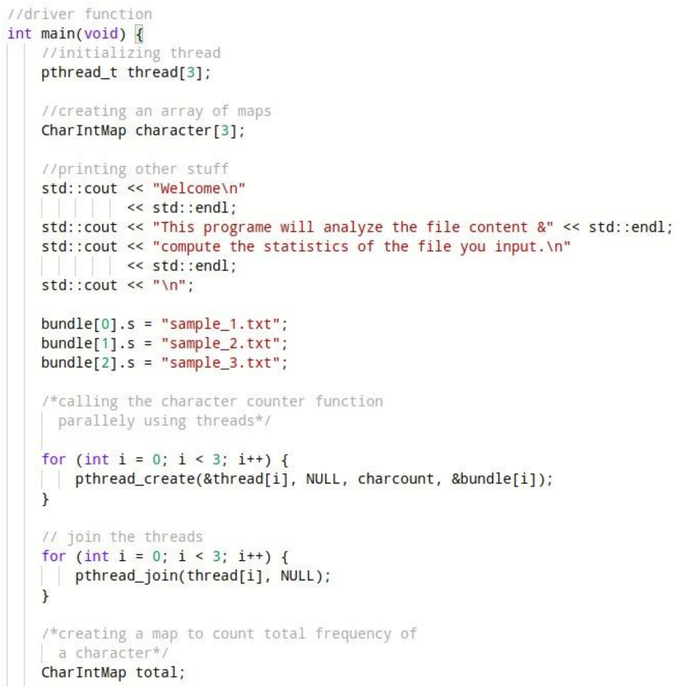
```
1. Type cast the function arguments from void to structure package.
2. Open file passed through function argument and store the return value in an integer variable.
3. Check if the integer variable is greater than 0
    a. If greater than zero, print error in opening file
    b. Else, continue
4. Create a buffer
5. read the opened file into buffer byte by byte until the end of file.
    a. If end of file is reached, break out of loop
    b. Else, map the character key and update the corresponding value.
```

Algorithm for main driver function:

```
1. Create an array of character integer map
2. Create an array of threads
3. Give the file names to the string variable of each bundle (object of the structure)
4. Print the introductory note
5. Run a loop to execute each thread
    a. call the pthread_create function to create a thread bu passing suitable arguments i.e.
        1. the address of each index of the thread array
        2. NULL
```

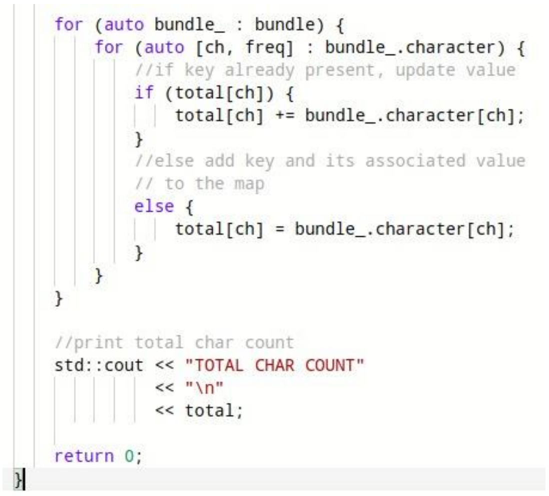
        3. the charcount function and
        4. the address of each index of the arrayed bundle.
4. run a loop to terminate the thread after execution is completed.
        a. call the pthread_join function with arguments i.e. thread of each index and Null value since the function doesn’t return any value.
4. create another character integer map to calculate total frequency
5. run a loop for each character in function charcount()
        a. run another loop for each [key, value] pair in charcount()
        b. check if key for each character in the file exists.
                i. if it exists, update the value corresponding to it
                ii. else, add the [key, value] to the total map.
8. Print out the individual frequencies for each file and then the total frequency of all the files.

Source code:



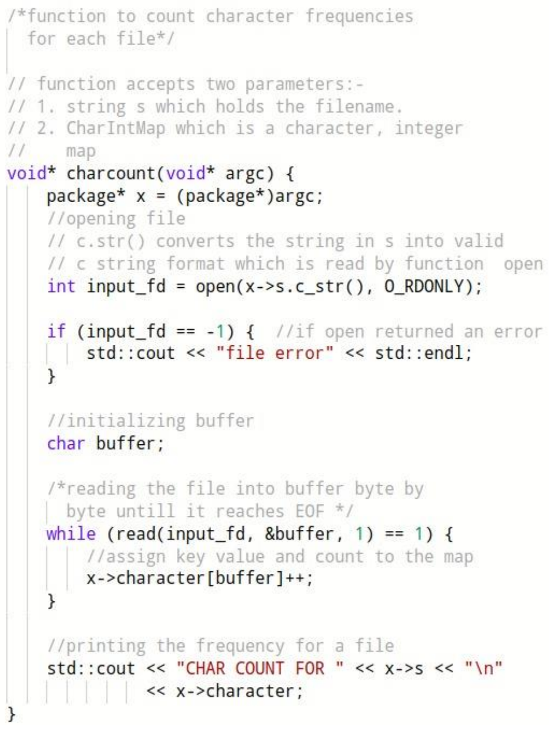
```
//driver function
int main(void) {
    //initializing thread
    pthread_t thread[3];

    //creating an array of maps
    CharIntMap character[3];

    //printing other stuff
    std::cout << "Welcome\n"
              << std::endl;
    std::cout << "This programe will analyze the file content &" << std::endl;
    std::cout << "compute the statistics of the file you input.\n"
              << std::endl;
    std::cout << "\n";

    bundle[0].s = "sample_1.txt";
    bundle[1].s = "sample_2.txt";
    bundle[2].s = "sample_3.txt";

    /*calling the character counter function
      parallely using threads*/

    for (int i = 0; i < 3; i++) {
        pthread_create(&thread[i], NULL, charcount, &bundle[i]);
    }

    // join the threads
    for (int i = 0; i < 3; i++) {
        pthread_join(thread[i], NULL);
    }

    /*creating a map to count total frequency of
      a character*/
    CharIntMap total;
```

Figure 11.a: source code for the driver function



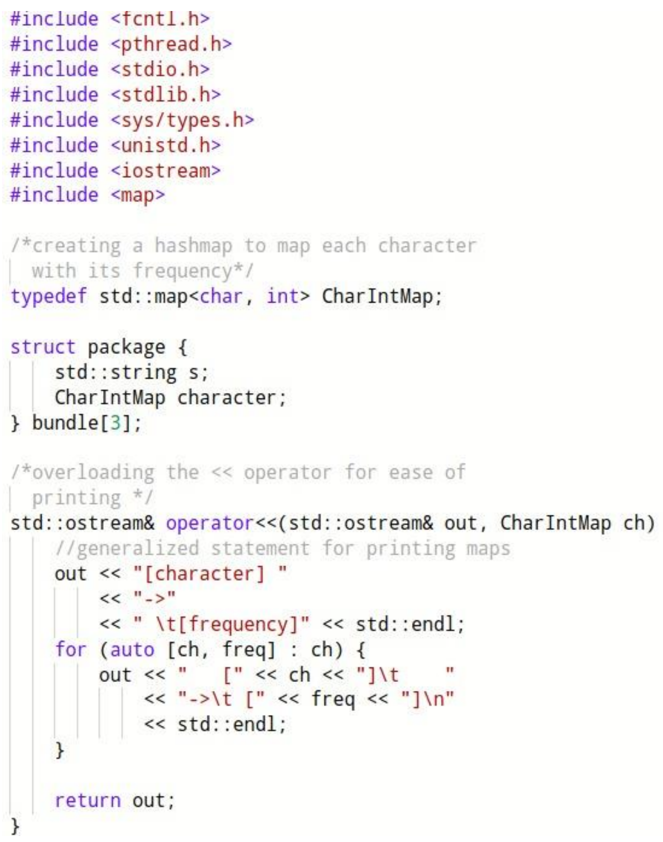
```
    for (auto bundle_ : bundle) {
        for (auto [ch, freq] : bundle_.character) {
            //if key already present, update value
            if (total[ch]) {
                total[ch] += bundle_.character[ch];
            }
            //else add key and its associated value
            // to the map
            else {
                total[ch] = bundle_.character[ch];
            }
        }
    }

    //print total char count
    std::cout << "TOTAL CHAR COUNT"
              << "\n"
              << total;

    return 0;
}
```

Figure 11.b: source code of the main driver function



```
/*function to count character frequencies
  for each file*/

// function accepts two parameters:-
// 1. string s which holds the filename.
// 2. CharIntMap which is a character, integer
//    map
void* charcount(void* argc) {
    package* x = (package*)argc;
    //opening file
    // c.str() converts the string in s into valid
    // c string format which is read by function  open
    int input_fd = open(x->s.c_str(), O_RDONLY);

    if (input_fd == -1) {  //if open returned an error
        std::cout << "file error" << std::endl;
    }

    //initializing buffer
    char buffer;

    /*reading the file into buffer byte by
      byte untill it reaches EOF */
    while (read(input_fd, &buffer, 1) == 1) {
        //assign key value and count to the map
        x->character[buffer]++;
    }

    //printing the frequency for a file
    std::cout << "CHAR COUNT FOR " << x->s << "\n"
              << x->character;
}
```

Figure 12: source code of the charcount function

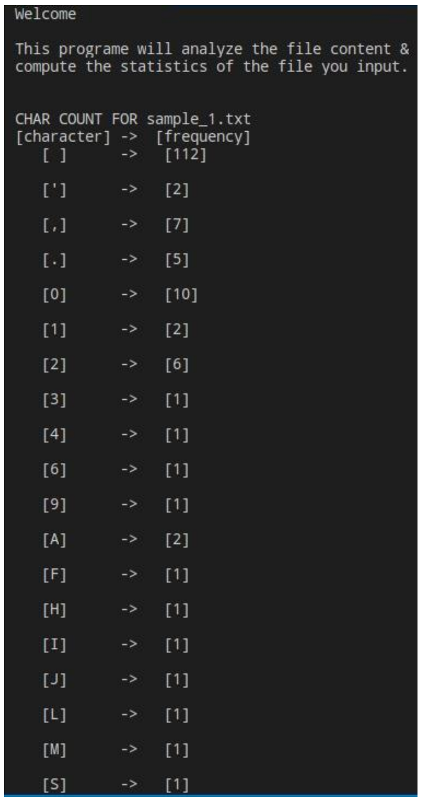


```cpp
#include <fcntl.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <iostream>
#include <map>

/*creating a hashmap to map each character
  with its frequency*/
typedef std::map<char, int> CharIntMap;

struct package {
    std::string s;
    CharIntMap character;
} bundle[3];

/*overloading the << operator for ease of
  printing */
std::ostream& operator<<(std::ostream& out, CharIntMap ch)
    //generalized statement for printing maps
    out << "[character] "
        << "->"
        << " \t[frequency]" << std::endl;
    for (auto [ch, freq] : ch) {
        out << "    [" << ch << "]\t    "
            << "->\t [" << freq << "]\n"
            << std::endl;
    }

    return out;
}
```

Figure 13:source code for the structure and the << operator overload

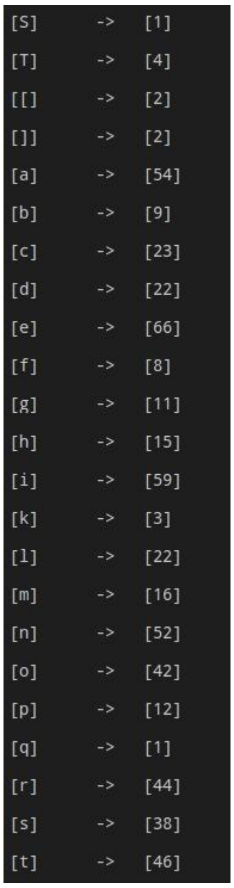
In order to run this source code, we took 3 sample files which are shown in the appendix.
This program reads every character in the sample file and prints the frequency of every character in this file. Also, the main function calculates the total frequency of every character in all the 3 files.

OUTPUT :



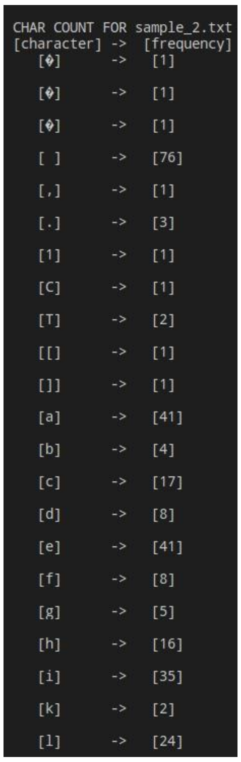
```
Welcome

This programe will analyze the file content &
compute the statistics of the file you input.


CHAR COUNT FOR sample_1.txt
[character] ->  [frequency]
   [ ]      ->   [112]

   [']      ->   [2]

   [,]      ->   [7]

   [.]      ->   [5]

   [0]      ->   [10]

   [1]      ->   [2]

   [2]      ->   [6]

   [3]      ->   [1]

   [4]      ->   [1]

   [6]      ->   [1]

   [9]      ->   [1]

   [A]      ->   [2]

   [F]      ->   [1]

   [H]      ->   [1]

   [I]      ->   [1]

   [J]      ->   [1]

   [L]      ->   [1]

   [M]      ->   [1]

   [S]      ->   [1]
```

Figure 14:output



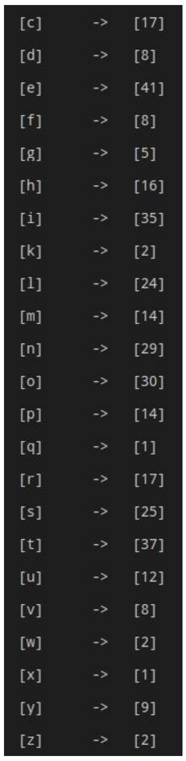
```
[S]      ->   [1]
[T]      ->   [4]
[[]      ->   [2]
[]]      ->   [2]
[a]      ->   [54]
[b]      ->   [9]
[c]      ->   [23]
[d]      ->   [22]
[e]      ->   [66]
[f]      ->   [8]
[g]      ->   [11]
[h]      ->   [15]
[i]      ->   [59]
[k]      ->   [3]
[l]      ->   [22]
[m]      ->   [16]
[n]      ->   [52]
[o]      ->   [42]
[p]      ->   [12]
[q]      ->   [1]
[r]      ->   [44]
[s]      ->   [38]
[t]      ->   [46]
```

Figure 15:output of sample 1 continued



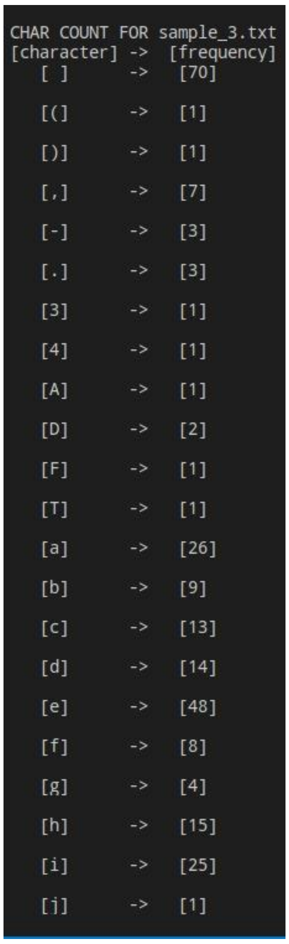
```
CHAR COUNT FOR sample_2.txt
[character] ->  [frequency]
   [�]      ->    [1]
   [�]      ->    [1]
   [�]      ->    [1]
   [ ]      ->    [76]
   [,]      ->    [1]
   [.]      ->    [3]
   [1]      ->    [1]
   [C]      ->    [1]
   [T]      ->    [2]
   [[]      ->    [1]
   []]      ->    [1]
   [a]      ->    [41]
   [b]      ->    [4]
   [c]      ->    [17]
   [d]      ->    [8]
   [e]      ->    [41]
   [f]      ->    [8]
   [g]      ->    [5]
   [h]      ->    [16]
   [i]      ->    [35]
   [k]      ->    [2]
   [l]      ->    [24]
```

Figure 16:output for sample 2



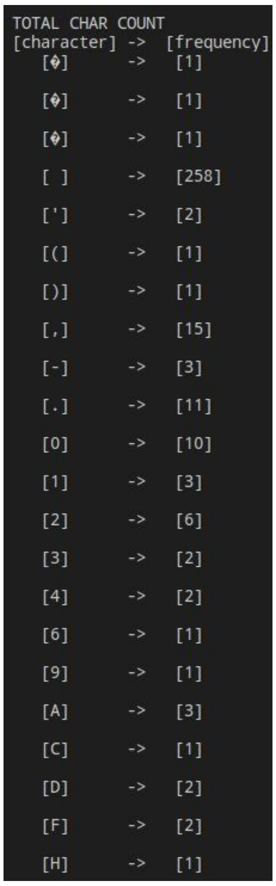
```
[c]      ->  [17]
[d]      ->  [8]
[e]      ->  [41]
[f]      ->  [8]
[g]      ->  [5]
[h]      ->  [16]
[i]      ->  [35]
[k]      ->  [2]
[l]      ->  [24]
[m]      ->  [14]
[n]      ->  [29]
[o]      ->  [30]
[p]      ->  [14]
[q]      ->  [1]
[r]      ->  [17]
[s]      ->  [25]
[t]      ->  [37]
[u]      ->  [12]
[v]      ->  [8]
[w]      ->  [2]
[x]      ->  [1]
[y]      ->  [9]
[z]      ->  [2]
```

Figure 17:output of sample 2 continued



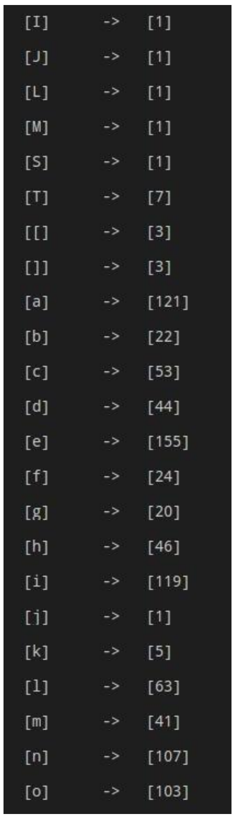
```
CHAR COUNT FOR sample_3.txt
[character] ->  [frequency]
   [ ]      ->   [70]

   [(]      ->   [1]

   [)]      ->   [1]

   [,]      ->   [7]

   [-]      ->   [3]

   [.]      ->   [3]

   [3]      ->   [1]

   [4]      ->   [1]

   [A]      ->   [1]

   [D]      ->   [2]

   [F]      ->   [1]

   [T]      ->   [1]

   [a]      ->   [26]

   [b]      ->   [9]

   [c]      ->   [13]

   [d]      ->   [14]

   [e]      ->   [48]

   [f]      ->   [8]

   [g]      ->   [4]

   [h]      ->   [15]

   [i]      ->   [25]

   [j]      ->   [1]
```

Figure 18:output for sample 3



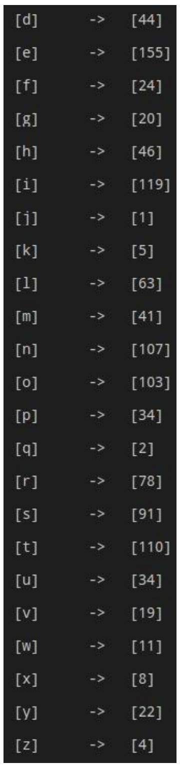
```
TOTAL CHAR COUNT
[character] ->  [frequency]
   [�]       ->    [1]

   [�]       ->    [1]

   [�]       ->    [1]

   [ ]       ->    [258]

   [']       ->    [2]

   [(]       ->    [1]

   [)]       ->    [1]

   [,]       ->    [15]

   [-]       ->    [3]

   [.]       ->    [11]

   [0]       ->    [10]

   [1]       ->    [3]

   [2]       ->    [6]

   [3]       ->    [2]

   [4]       ->    [2]

   [6]       ->    [1]

   [9]       ->    [1]

   [A]       ->    [3]

   [C]       ->    [1]

   [D]       ->    [2]

   [F]       ->    [2]

   [H]       ->    [1]
```

Figure 19:output for total char count in all three files



```
[I]        ->    [1]

[J]        ->    [1]

[L]        ->    [1]

[M]        ->    [1]

[S]        ->    [1]

[T]        ->    [7]

[[]        ->    [3]

[]]        ->    [3]

[a]        ->    [121]

[b]        ->    [22]

[c]        ->    [53]

[d]        ->    [44]

[e]        ->    [155]

[f]        ->    [24]

[g]        ->    [20]

[h]        ->    [46]

[i]        ->    [119]

[j]        ->    [1]

[k]        ->    [5]

[l]        ->    [63]

[m]        ->    [41]

[n]        ->    [107]

[o]        ->    [103]
```

Figure 20:output for total char count continued

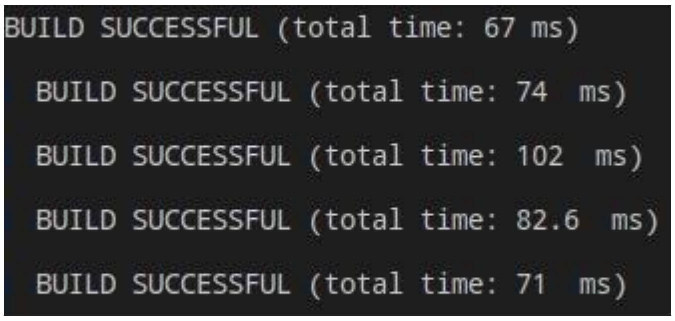

```
[d]     ->  [44]
[e]     ->  [155]
[f]     ->  [24]
[g]     ->  [20]
[h]     ->  [46]
[i]     ->  [119]
[j]     ->  [1]
[k]     ->  [5]
[l]     ->  [63]
[m]     ->  [41]
[n]     ->  [107]
[o]     ->  [103]
[p]     ->  [34]
[q]     ->  [2]
[r]     ->  [78]
[s]     ->  [91]
[t]     ->  [110]
[u]     ->  [34]
[v]     ->  [19]
[w]     ->  [11]
[x]     ->  [8]
[y]     ->  [22]
[z]     ->  [4]
```
Figure 21:final output for total char count

the output of this question and in question 1.1 are identical since the were fed the same sample files and the output indentation is also similar. Only difference is in the source code as it follows a different approach.

1.3 Comparison of execution time and analysis:

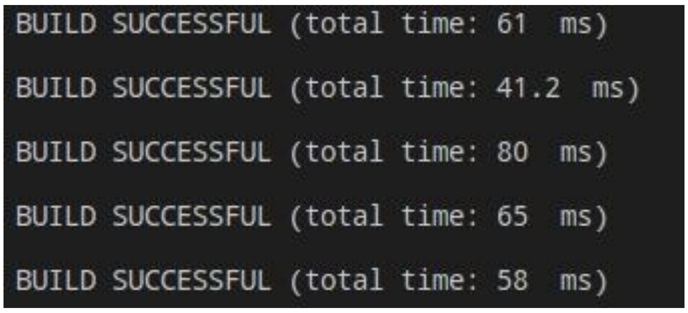
To compare and analyze the execution time of both the approaches, each subsequent approach was implemented to a minimum of 5 times. With each iteration containing 3 files and their execution time was noted down. This execution time depends on several factors like the background processes running on the system. Since, the idle execution time of small sample files will be in **nanoseconds**, which is not an optimal case of comparison because of its minimal scale, the sample files were chosen to be large (nearly 10 mb) to scale the execution time to **micro seconds**.

To compare the execution time, there shall be no user inputs of any kind and the IDE will be the only software or application running at the time of execution, just to avoid any exterior disturbance, which may alter the values of execution time. Average of the 5 runs shall be consider to make the decision. Below the are results of execution:



```
BUILD SUCCESSFUL (total time: 67 ms)
  BUILD SUCCESSFUL (total time: 74  ms)
  BUILD SUCCESSFUL (total time: 102  ms)
  BUILD SUCCESSFUL (total time: 82.6  ms)
  BUILD SUCCESSFUL (total time: 71  ms)
```

Figure 22: successive iterations of the sequential approach



```
BUILD SUCCESSFUL (total time: 61  ms)
BUILD SUCCESSFUL (total time: 41.2  ms)
BUILD SUCCESSFUL (total time: 80  ms)
BUILD SUCCESSFUL (total time: 65  ms)
BUILD SUCCESSFUL (total time: 58  ms)
```

Figure 23: successive iterations of the multithreaded approach

NOTE: In order to compare both the approaches, large sample size of the file is used with an average size of each file to approximately 10 to 15 mb. And as the program computes the frequency of each individual character, the output is not shown here. Only the execution time for each execution is shown.

Table 1: tabulated execution time of each iteration for each approach

Program approach	1st iteration	2nd iteration	3rd iteration	4th iteration	5th iteration	Avg. exe. time
Sequential	67	74	102	82.6	71	79.32
Multithreaded	61	41.2	80	65	58	61.04

Looking at the table, it is very evident that multithreaded program takes less execution time. We know that execution time is inversely proportional to the CPU performance. So, lower the execution time, better is the CPU performance. Hence, it can be suggested that multithreaded approach is better than sequential approach in terms of better CPU performance.

Solution to Question No. 2:

2.1 Number of page faults that occur when FIFO, LRU and Optimal page replacement algorithms are used respectively.

Introduction: -

In this question we are going to deal with page replacement algorithm so before dealing with page replacement algorithm we going to discuss about page replacement. Page replacement is allotting of pages in the memory so memory has frame size which is responsible to hold processes at a time thus, page replacement will check the page which is need to be replaced when there is new page to allot in a memory or at a same time there was no free frame. All of this can be done by using different types of algorithm that are mentioned below.

- FIFO
- LRU
- Optimal

So, in 1st question we are going to find page fault by drawing table of three different algorithms and with the help of table we calculate page fault by using formula

Total number of page fault is = total physical page – page hit

Here, page hit will be mark as (*)

FIFO: -

Table 1.0 Framing string of physical pages by using FIFO algorithm

Frames	0	1	2	3	2	3	0	4	5	2	3	1	4	3	2	6	3	2	1	2
0	0	0	0	3	3	3	3	3	5	5	5	1	1	1	1	6	6	6	6	2
1		1	1	1	1	1	0	0	0	2	2	2	4	4	4	4	3	3	3	3
2			2	2	2	2	2	4	4	4	3	3	3	3	2	2	2	2	1	1
					*	*								*				*		

In table 1 we are going to allotting the memory for the string of physical pages by using FIFO algorithm. In FIFO algorithm i.e. first in first out we are going to replace the oldest page in the memory by the next string of physical pages. If there is page hit occur for any string then same allocation will retain and it will be denoted by star (*) mark.

Since, table 1.0 is not clearly show how the algorithm work so we goanna made table by keeping stack in mind so, At first **0** will come and will get loaded because the frame was free then **1** will come so I just push **0** in front of the queue (according to my table pushes below) so pushing will be carry on until the frame is free i.e. till **2** after that when **3** will come then page in the front of the queue is selected for removal after that **2** will come it is known as hit and whatever will hit will be the first one in the next column and other will be coming down in same sequence. So, that it will become easy to replace page as per FIFO.

Note: - All the string at frame 2 will be replaced in the next step because it was in the first position in the queue until the hit will not occur.

Table 2.1 Framing string of physical pages by using FIFO algorithm.

Frames	0	1	2	3	2	3	0	4	5	2	3	1	4	3	2	6	3	2	1	2
0	0	1	2	3	3	3	0	4	5	2	3	1	4	4	2	6	3	3	1	2
1		0	1	2	2	2	3	0	4	5	2	3	1	1	4	2	6	6	3	1
2			0	1	1	1	2	3	0	4	5	2	3	3	1	4	2	2	6	3
					*	*								*				*		

Total number of page fault is = total physical page – page hit

= 20 – 4 = **16**

LRU: -

Table 3.0 Framing string of physical pages by using LRU algorithm.

Frames	0	1	2	3	2	3	0	4	5	2	3	1	4	3	2	6	3	2	1	2
0	0	0	0	3	3	**3**	3	3	5	5	5	1	1	1	2	2	2	**2**	2	**2**
1		1	1	1	1	1	0	0	0	2	2	2	4	4	4	6	6	6	1	1
2			2	2	**2**	2	2	4	4	4	3	3	3	**3**	3	3	**3**	3	3	3
					*	*								*			*	*		*

In table 2.0 we are going to allotting the memory for the string of physical pages by using LRU algorithm. In LRU algorithm i.e. least recent used we are going to replace the page which has not been used for the longest time is replaced. If there is page hit occur for any string then same allocation will retain and with that, we have to refresh the count of the string that are being hit i.e. we are not going to count previous **2's** before string **2** and it will be denoted by star (*) mark.

Again table 2.0 is not clear so we goanna made table by keeping stack in mind so,

At first **0** come and get loaded since frame is free same for **1** and **2** then for **3**, frame is not free so we need to replace string as per LRU so **0** i.e. least recent used string will be replaced by **3.**

Table 2.0 Framing string of physical pages by using LRU algorithm

Frames	0	1	2	3	2	3	0	4	5	2	3	1	4	3	2	6	3	2	1	2
0	0	1	2	3	2	**3**	0	4	5	2	3	1	4	3	2	6	3	**2**	1	**2**
1		0	1	2	3	2	3	0	4	5	2	3	1	4	3	2	6	3	2	1
2			0	1	**1**	1	2	3	0	4	5	2	3	**1**	4	3	**2**	6	3	3
					*	*								*			*	*		*

Total number of page fault is = total physical page – page hit

= 20 – 6 = **14**

Optimal: -

Table 4.0 Framing string of physical pages by using Optimal algorithm

Frames	0	1	2	3	2	3	0	4	5	2	3	1	4	3	2	6	3	2	1	2
0	0	0	0	0	0	0	0	4	5	5	5	1	4	4	4	6	6	6	1	1
1		1	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
2			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
					*	*	*			*	*			*	*		*	*		*

In table 3.0 we are going to allotting the memory for the string of physical pages by using Optimal algorithm. In Optimal algorithm we are going to replace the page which would not be used for the longest duration of time in the future. If there is page hit occur for any string then same allocation will retain and with that and it will be denoted by star (*) mark.

Total number of page fault is = total physical page – page hit

= 20 – 10 = **10**

2.2 Diagram of the probability density function of distance strings based on LRU:

In this question we are going to draw the diagram of the probability density function of distance strings based on LRU algorithm:

Table 5 Framing string of physical pages by using LRU algorithm

	0	1	2	3	2	3	0	4	5	2	3	1	4	3	2	6	3	2	1	2
	0	1	2	3	2	3	0	4	5	2	3	1	4	3	2	6	3	2	1	2
		0	1	2	3	2	3	0	4	5	2	3	1	4	3	2	6	3	2	1
			0	1	1	1	2	3	0	4	5	2	3	1	4	3	2	6	3	3
				0	0	0	1	2	3	0	4	5	2	2	1	4	4	4	6	6
								1	2	3	0	4	5	5	5	1	1	1	4	4
									1	1	1	0	0	0	0	5	5	5	5	5

																0	0	0	0	0
String distance	X	X	X	X	2	2	4	X	X	5	5	6	5	3	4	X	3	3	5	2
					*	*	*			*	*			*	*		*	*		*

Here **X** indicates the distance as infinity and others string (given in question) distances are clearly calculated by counting the string diagonally and if it is occurring first time then the distance is supposed to be infinity (denoted by X).

Now, as per question we have to find probability density function of distance strings based on LRU algorithm. So, for finding that we use

Probability of string = $\frac{\text{string distance}}{\text{total number of refrence string}}$

Probability of string 1= p (1) = **0**

Probability of string 2= p (2) = $\frac{3}{20}$ = **0.15**

Probability of string 3= p (3) = $\frac{3}{20}$ = **0.15**

Probability of string 4= p (4) = $\frac{2}{20}$ = **0.1**

Probability of string 5= p (5) = $\frac{4}{20}$ = **0.2**

Probability of string 6= p (6) = $\frac{1}{20}$ = **0.05**

Probability of infinity= p(infinity) = $\frac{7}{20}$ = **0.35**

This can be visualized graphically using matlab as shown in figure below:

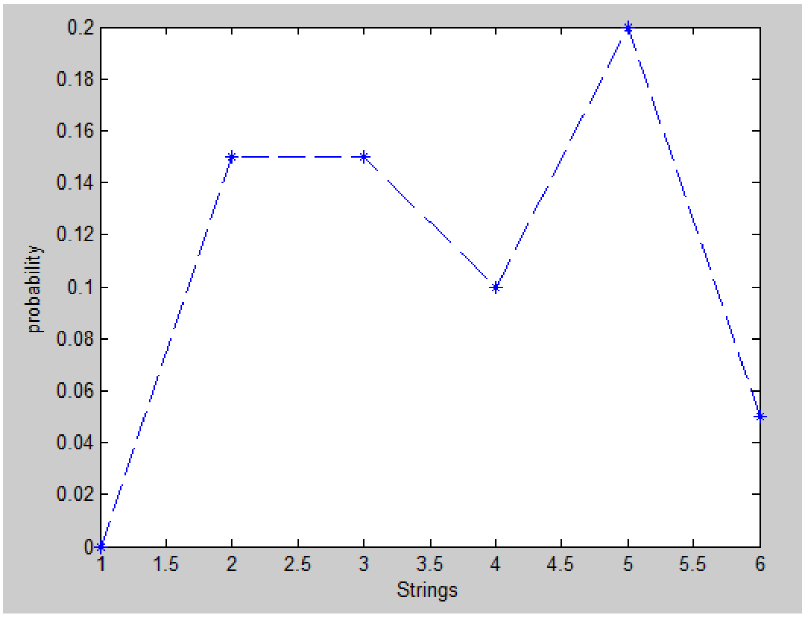
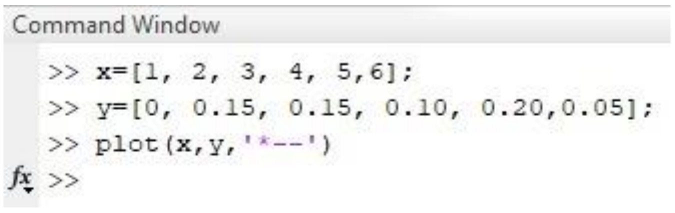


Figure: Graph of probability density function

The above graph is a plot for probability density function vs distance strings. Here, the string distance is stored in x vector. Infinite string distance was not considered. Hence, only strings 1 to 6 were considered and their corresponding probability density was stored in the y array. The plot function in matlab was used to plot both the values in the graph.

MATLAB CODE:



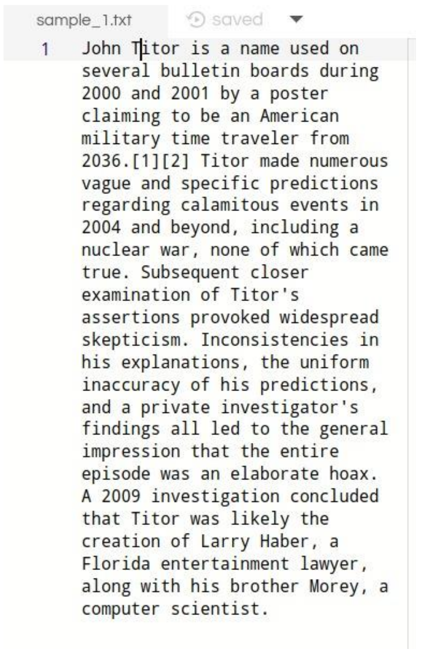
```
>> x=[1, 2, 3, 4, 5,6];
>> y=[0, 0.15, 0.15, 0.10, 0.20,0.05];
>> plot(x,y,'*--')
>>
```

2.3 Recommendations of an optimal number of physical page frames appropriate for the given string of accesses.

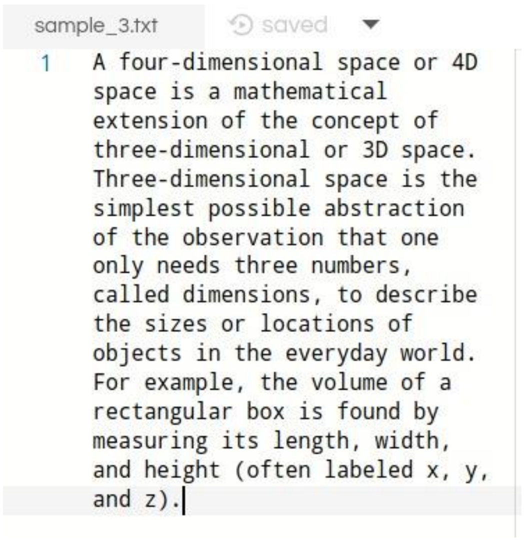
Looking at the probability density function, we can see that the distance string 5 has the maximum probability (probability of infinite distance string is high, but it is practically not possible to have infinite physical page frames).

Therefore, having 5 page frames will reduce the number of page faults for the given set of string reference. Thus 5 is the optimal number of physical page frames.

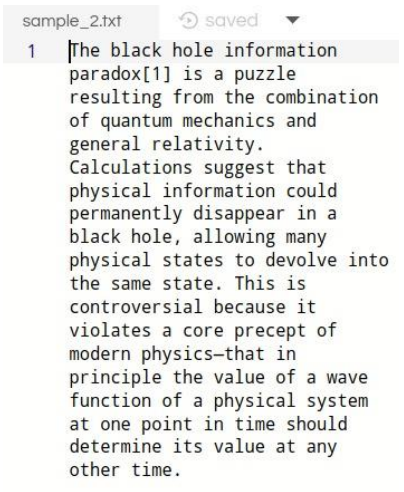
The three sample files chosen for question 1.1 and 1.2


sample_1.txt saved

```
John Titor is a name used on
several bulletin boards during
2000 and 2001 by a poster
claiming to be an American
military time traveler from
2036.[1][2] Titor made numerous
vague and specific predictions
regarding calamitous events in
2004 and beyond, including a
nuclear war, none of which came
true. Subsequent closer
examination of Titor's
assertions provoked widespread
skepticism. Inconsistencies in
his explanations, the uniform
inaccuracy of his predictions,
and a private investigator's
findings all led to the general
impression that the entire
episode was an elaborate hoax.
A 2009 investigation concluded
that Titor was likely the
creation of Larry Haber, a
Florida entertainment lawyer,
along with his brother Morey, a
computer scientist.
```


sample_3.txt saved

```
A four-dimensional space or 4D
space is a mathematical
extension of the concept of
three-dimensional or 3D space.
Three-dimensional space is the
simplest possible abstraction
of the observation that one
only needs three numbers,
called dimensions, to describe
the sizes or locations of
objects in the everyday world.
For example, the volume of a
rectangular box is found by
measuring its length, width,
and height (often labeled x, y,
and z).
```


sample_2.txt saved

```
The black hole information
paradox[1] is a puzzle
resulting from the combination
of quantum mechanics and
general relativity.
Calculations suggest that
physical information could
permanently disappear in a
black hole, allowing many
physical states to devolve into
the same state. This is
controversial because it
violates a core precept of
modern physics—that in
principle the value of a wave
function of a physical system
at one point in time should
determine its value at any
other time.
```

1. https://www.geeksforgeeks.org/multithreading-c-2/
2. https://www.geeksforgeeks.org/print-system-time-c-3-different-ways/
3. https://www.geeksforgeeks.org/page-replacement-algorithms-in-operating-systems/amp/
4. https://www.geeksforgeeks.org/optimal-page-replacement-algorithm/amp/
5. https://www.geeksforgeeks.org/program-for-least-recently-used-lru-page-replacement-algorithm/amp/