

Group Member’s

1. Muhammad Furqan Waseem (28)
2. Muzamil Ali Kazmi (30)
3. Hamza Ilyas (32)
4. Talha Sheikh (44)

Project Title

* First Come First Serve Algorithm

Resource Person

Engineer Noor-ul-Sabah

Contents

[**Summary** 3](file:///F:\Operting%20System\Final%20Project\Final%20Documentation.docx#_Toc78019344)

[**Introduction** 3](file:///F:\Operting%20System\Final%20Project\Final%20Documentation.docx#_Toc78019345)

[**Scheduler** 3](#_Toc78019346)

[**Algorithm** 3](#_Toc78019347)

[**FCFS (First Come First Serve)** 3](file:///F:\Operting%20System\Final%20Project\Final%20Documentation.docx#_Toc78019348)

[**Working** 4](#_Toc78019349)

[**Algorithm of FCFS** 4](#_Toc78019350)

[**Project** 4](file:///F:\Operting%20System\Final%20Project\Final%20Documentation.docx#_Toc78019351)

[**Class Diagram** 5](#_Toc78019352)

[**Libraries** 5](file:///F:\Operting%20System\Final%20Project\Final%20Documentation.docx#_Toc78019353)

[**iostream** 6](#_Toc78019354)

[**fstream** 6](#_Toc78019355)

[**chrono** 6](#_Toc78019356)

[**stdlib.h** 6](#_Toc78019357)

[**Algorithm** 6](#_Toc78019358)

[**Student** 6](file:///F:\Operting%20System\Final%20Project\Final%20Documentation.docx#_Toc78019359)

[**Insert Student Data** 7](#_Toc78019360)

[**Code** 7](#_Toc78019361)

[**Read Student Data** 8](#_Toc78019362)

[**Code** 8](#_Toc78019363)

[**Delete Student Data** 9](#_Toc78019364)

[**Code** 9](#_Toc78019365)

[**Search Student Data** 12](#_Toc78019366)

[**Code** 12](#_Toc78019367)

[**Number of Lines** 14](#_Toc78019368)

[**Code** 14](#_Toc78019369)

[**Get Flag** 14](#_Toc78019370)

[**Code** 14](#_Toc78019371)

[**Algorithm** 14](file:///F:\Operting%20System\Final%20Project\Final%20Documentation.docx#_Toc78019372)

[**Get burst time and processes** 15](#_Toc78019373)

[**Code** 15](#_Toc78019374)

[**Get Length** 15](#_Toc78019375)

[**Code** 15](#_Toc78019376)

[**FCFS working** 15](#_Toc78019377)

[**Code** 15](#_Toc78019378)

[**Display** 15](#_Toc78019379)

[**Code** 16](#_Toc78019380)

[**Int main()** 17](file:///F:\Operting%20System\Final%20Project\Final%20Documentation.docx#_Toc78019381)

[**Code** 17](#_Toc78019383)

[**Reference** 21](#_Toc78019384)

# **Summary**

**We are working on this project since 16th of July, 2021. This project mainly based on the working of the FCFS (First Come First Serve) scheduler algorithm. There are total 4 processes in this project which are insert, read, delete and search. This project calculates the total time taken (time are calculated in milliseconds) by each process and tells the average waiting time and turn around time.**

# **Introduction**

## **Scheduler**

Scheduler is the system software which handles the process handling. The main purpose of the scheduler is to select which process to run. There are three types of schedulers.

1. Short term scheduler
2. Middle term scheduler
3. Long term scheduler

## **Algorithm**

There are multiple types of scheduler algorithms.

1. FCFS (First Come First Serve)
2. SJN (Short Job Next) Scheduling
3. Priority Scheduling
4. Shortest Remaining Time
5. RR (Round Robin) Scheduling
6. Multiple Level Queues Scheduling

# **FCFS (First Come First Serve)**

This algorithm will create the queue and store the processes in the queue than executes them turn wise. This will store the starting of the process. This is the simplest and easiest scheduler algorithm. See table (3.3) for the understanding of FCFS.

|  |  |  |  |
| --- | --- | --- | --- |
| **Processes** | **Burst Time** | **Waiting Time** | **Turn Around Time** |
| P1 | 33 | 0 | 33 |
| P2 | 5 | 33 | 38 |
| P3 | 1 | 38 | 39 |
| P4 | 55 | 39 | 94 |

Table (3.3) FCFS.

## **Working**

In this a queue is created and processes are stored in it. CPU calculates the burst time and waiting time of the first process is 0 than waiting time of all the processes are the sum of the waiting and burst time of the previous process. Turn around time is the sum of current burst and waiting time.

## **Algorithm of FCFS**

* Insert burst time of all processes
* The waiting time of first process is zero
* Find the waiting time of all the processes
* waitingTime[i] = burstTimep[i – 1] + waitingTime[i – 1]
* Find the turn Around Time

turnAroundTime[i] = waitingTime[i] + burstTimep[i]

* Find average waiting time

Average = totalWaitingTime/No.OfProcesses

* Find the average turn around Time

Average = totalTurnAroundTime/No.OfProcesses

# **Project**

Our project mainly based on the working of FCFS. We created two classes.

1. Student
2. Algorithm

From student class we take the processes and how much time is taken by each process and pass the time and process into the process array, burst time array and than calculate the waiting time, turn around time, average waiting time and average turn around time.

|  |
| --- |
| Algorithm |
| -processes[100] : string  -waitingTime[100] : double  -burstTime[100] : double  -turnAroundTime[100] : double  -length : int |
| +Algorithm() : void  +getBurstTimeAndProcess() : void  +fcfsWorking() : void  +display() : void  +getLength() : int |

## **Class Diagram**

|  |
| --- |
| **Student** |
| -read : ifstream  -write : ofstream  -name : string  - grade : string  -readData : string  -marks : float  -counter : int  -store : int  -flag : int  -rollNumber : int  -deleteCounter : int |
| +Student()  +insertStudentData() : void  +readStudentData() : void  +deleteStudentData() : void  +searchStudent() : void  +getFlag() : int  +numberOfLine() : int |

# **Libraries**

In this project we used following libraries

1. iostream
2. fstream
3. chrono
4. stdlib.h
5. Algorithum

# **iostream**

The C++ <iostream> header file declares a set of functions for standardInput**/**Output. It also defines I/O stream objects such as cin, cout e.t.c.

# **fstream**

This data type represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files. To perform file processing in C++, headerfiles <iostream> and <fstream> must be included in your c++ source file.

# **chrono**

It has two distinct objects–timepoint and duration. A timepoint as the name suggests represents a point in time whereas a duration represents an interval or span of time. The C++ library allows us to subtract two timepoints to get the interval of time passed in between. Using provided methods we can also convert this duration to appropriate units.The **std::chrono** provides us with three clocks with varying accuracy. The **high\_resolution\_clock** is the most accurate and hence it is used to measure execution time.

# **stdlib.h**

stdlib stands for standard library. This is used for the clearing screen.

## **Algorithm**

It is used for count function.

# **Student**

Student class contains 6 functions

1. Insert Student Data
2. Read Student Data
3. Delete Student Data
4. Search for Student Data
5. Number of lines
6. Get Flag

From student class functions we’ll get the time. The time is in the form of milliseconds.

## **Insert Student Data**

This function will take the student data as the input from the user and store it into the text file “studentData.txt”. This takes name, grade, marks and roll number as parameters. Figure 5.1

### **Code**

void insertStudentData(string Name , string Grade , float Marks , int RollNumber)

{

name = Name;

grade = Grade;

marks = Marks;

rollNumber = RollNumber;

write.open("studentData.txt" , ios::app);

write << name << endl;

write << grade << endl;

write << marks << endl;

write << rollNumber << endl;

write.close();

}

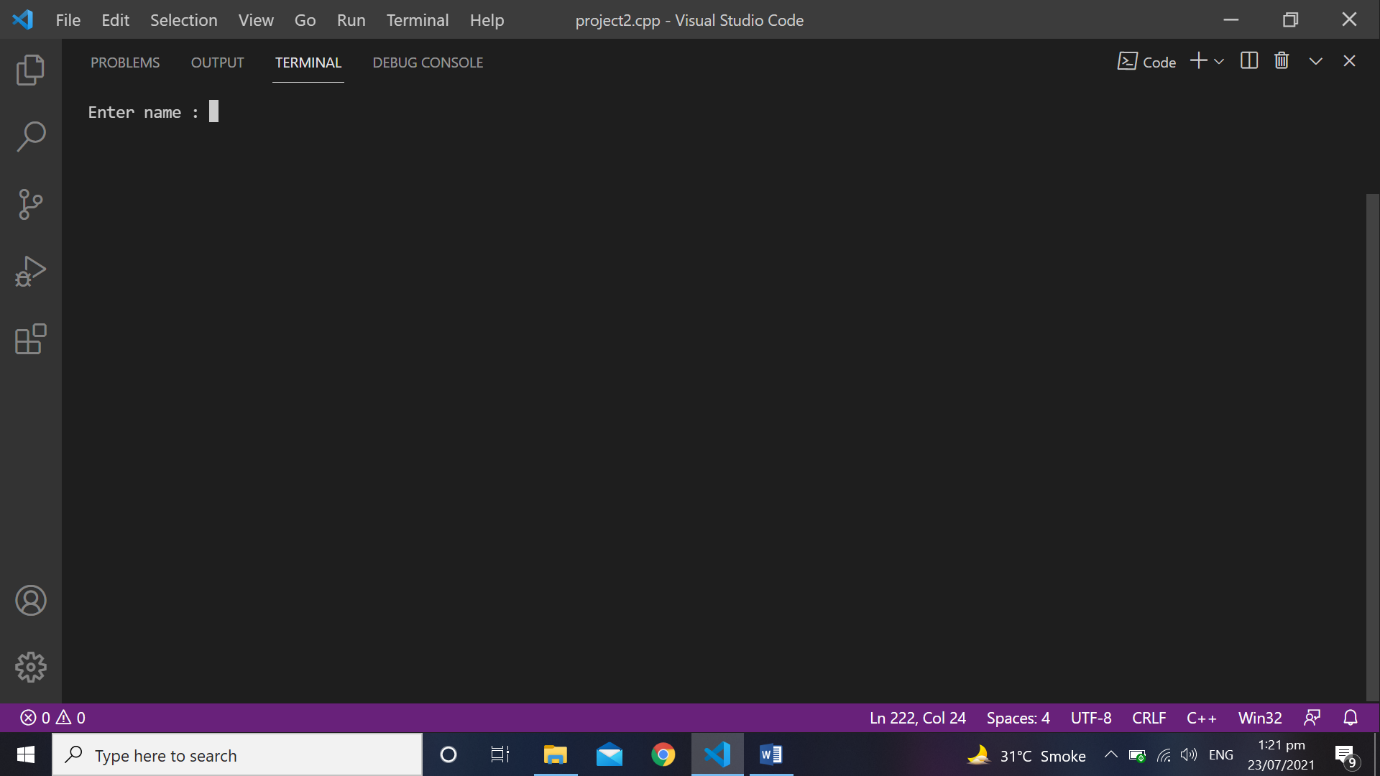


Figure 5.1(a)

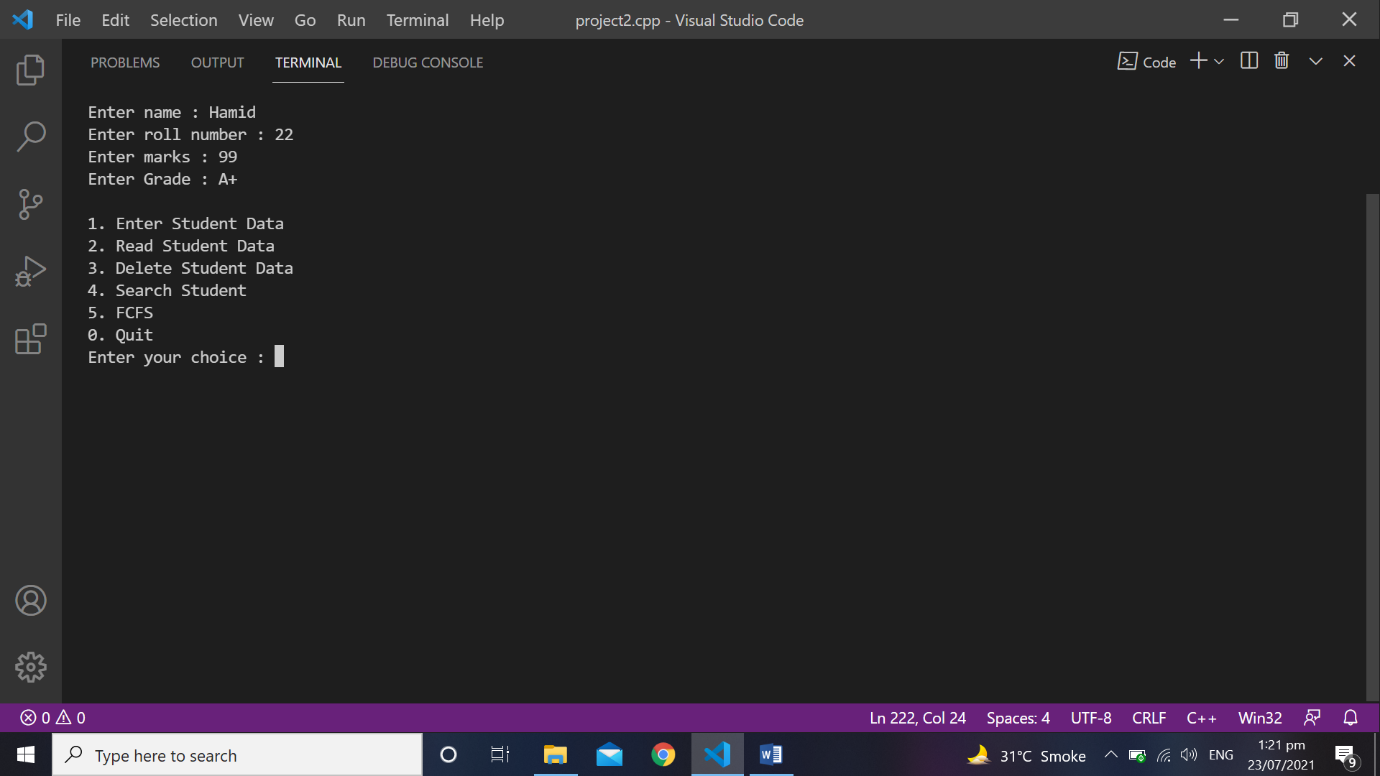


Figure 5.1(b)

## **Read Student Data**

This function will display the student data which is stored in the file “studentData.txt”. Figure 5.2

### **Code**

void readStudentData()

{

store = numberOfLine()/4;

if(store == 0)

{

cout << "File is empty" << endl;

return;

}

read.open("studentData.txt");

cout << "Name" << "\t" << "Roll Number" << "\t" << "Marks" << "\t" << "Grade" << endl;

for(int i = 1 ; i <= store ; i++)

{

read >> name;

read >> grade;

read >> marks;

read >> rollNumber;

cout << name << " \t " << rollNumber << " \t " << marks << " \t " << grade << endl;

}

read.close();

}

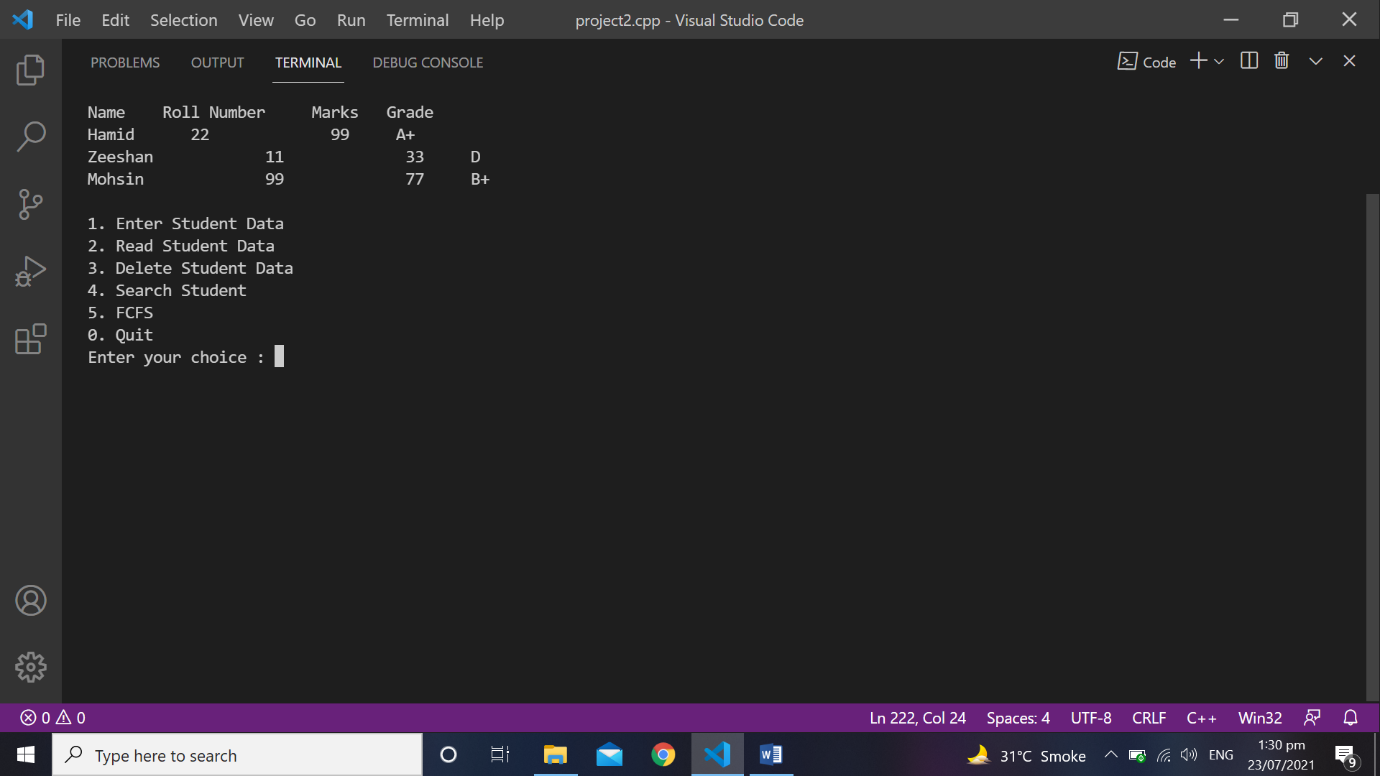


Figure 5.2

## **Delete Student Data**

This function will take the roll number of student from the user and check it in the file. If it exists than it will delete it from the file and set flag to “1” Figure 5.4(a). If it doesn’t exist than it will display “Roll number doesn’t exists” than it left flag equal to “0” Figure 5.4 (b). For deleting it will create the new file and copy all the other roll number in the new file “temp.txt” and rename it as again “studentdata.txt”.

### **Code**

void deleteStudentData(int RollNumber)

{

flag = 0;

store = numberOfLine()/4;

if(store == 0)

{

cout << "File is empty" << endl;

return;

}

read.open("studentData.txt");

write.open("temp.txt" , ios::app);

for(int i = 1 ; i <= store ; i++)

{

read >> name;

read >> grade;

read >> marks;

read >> rollNumber;

if(RollNumber == rollNumber)

{

cout << endl;

cout << name << endl;

cout << rollNumber << endl;

cout << marks << endl;

cout << grade << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* Above student record is deleted \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

flag = 1;

}

else

{

write << name << endl;

write << grade << endl;

write << marks << endl;

write << rollNumber << endl;

}

}

read.close();

write.close();

remove("studentData.txt");

rename("temp.txt" , "studentData.txt");

}

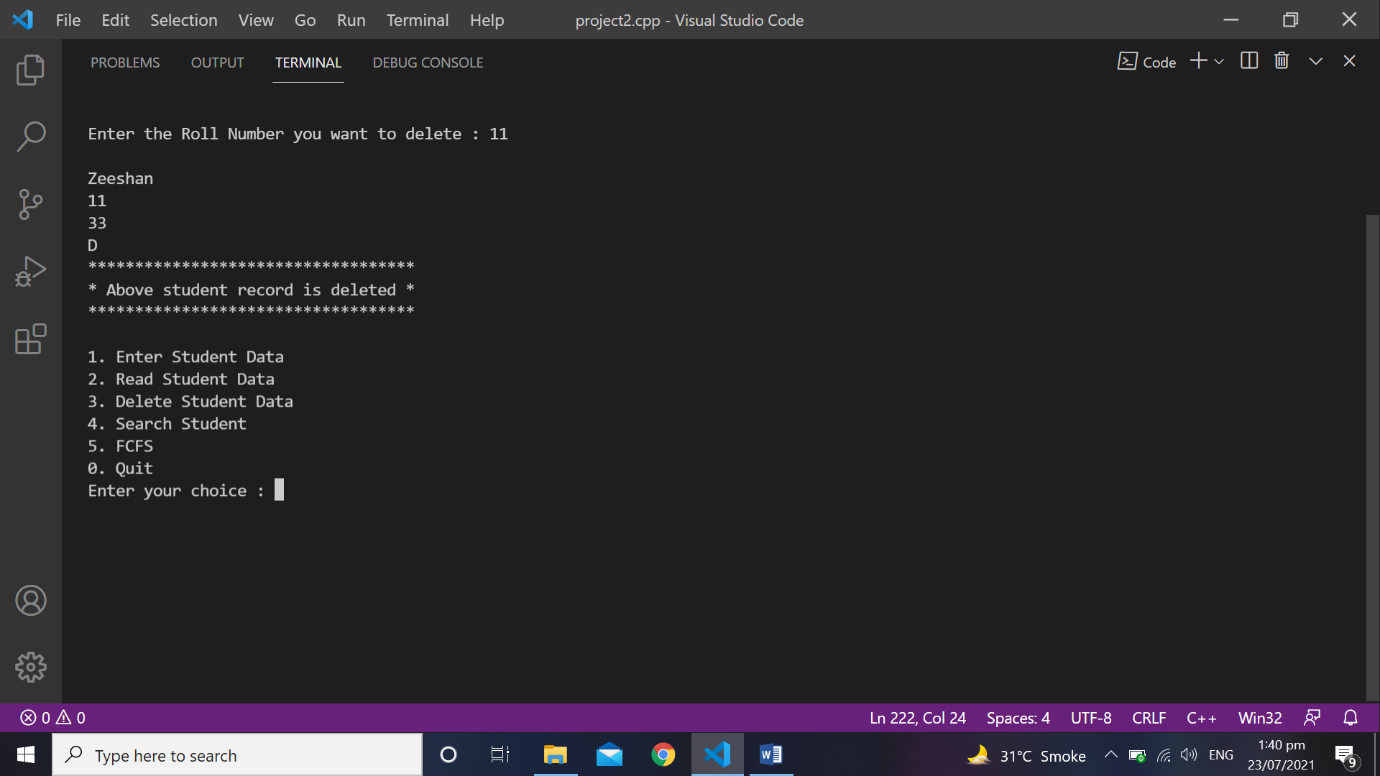


Figure 5.4(a)

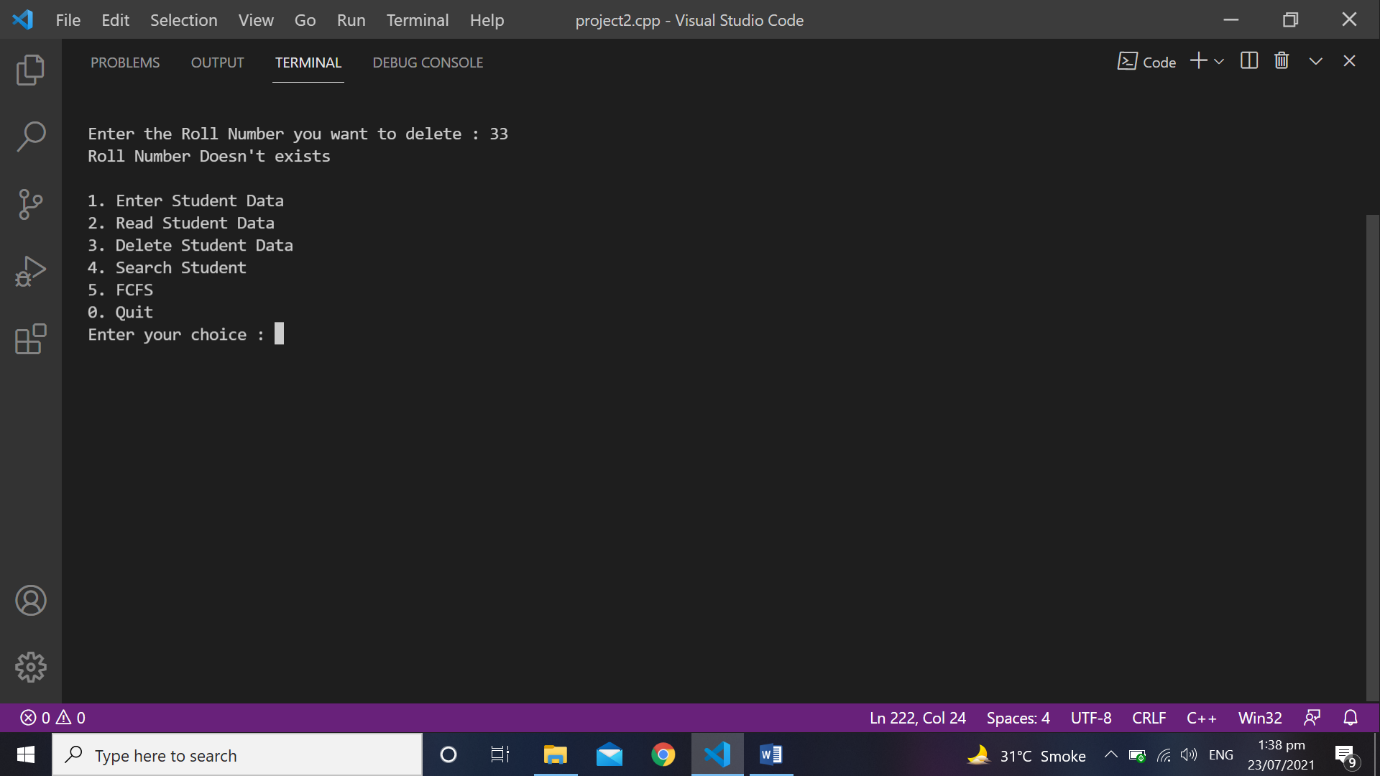


Figure 5.4(b)

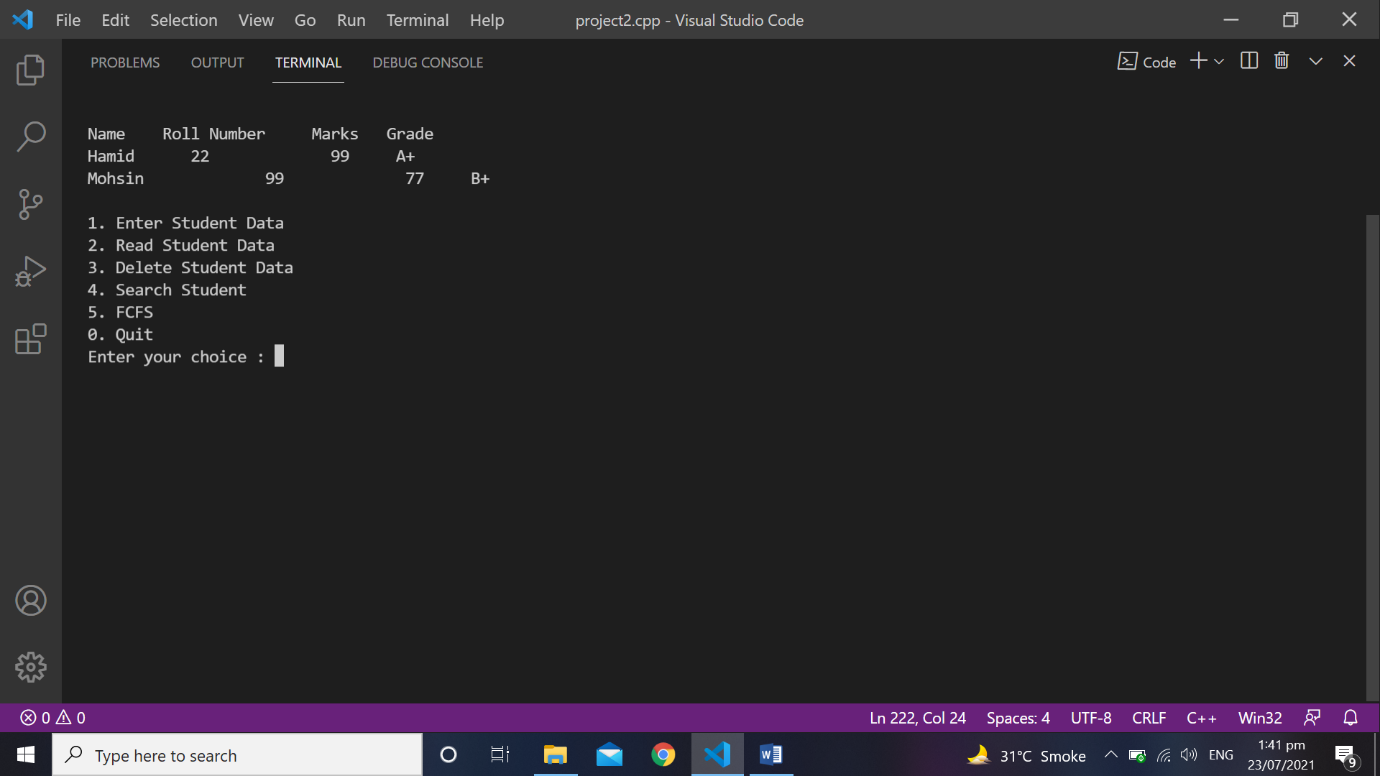


Figure 5.4(c) After deleting.

## **Search Student Data**

This function takes the roll number from the user and check it from the file. If it exists than it set flag to “1” Figure 5.5 (a). If it doesn’t exist than it will set the flag to “0” and display “Roll number doesn’t exist” Figure 5.5 (b).

### **Code**

void searchStudent(int index)

{

flag = 0;

read.open("studentData.txt");

while(!read.eof())

{

read >> name;

read >> grade;

read >> marks;

read >> rollNumber;

if(rollNumber == index)

{

cout << "Name : " << name<< endl;

cout << "Roll Number : " << rollNumber << endl;

cout << "Marks : " << marks << endl;

cout << "Grade : " << grade << endl;

flag = 1;

break;

}

}

read.close();

}

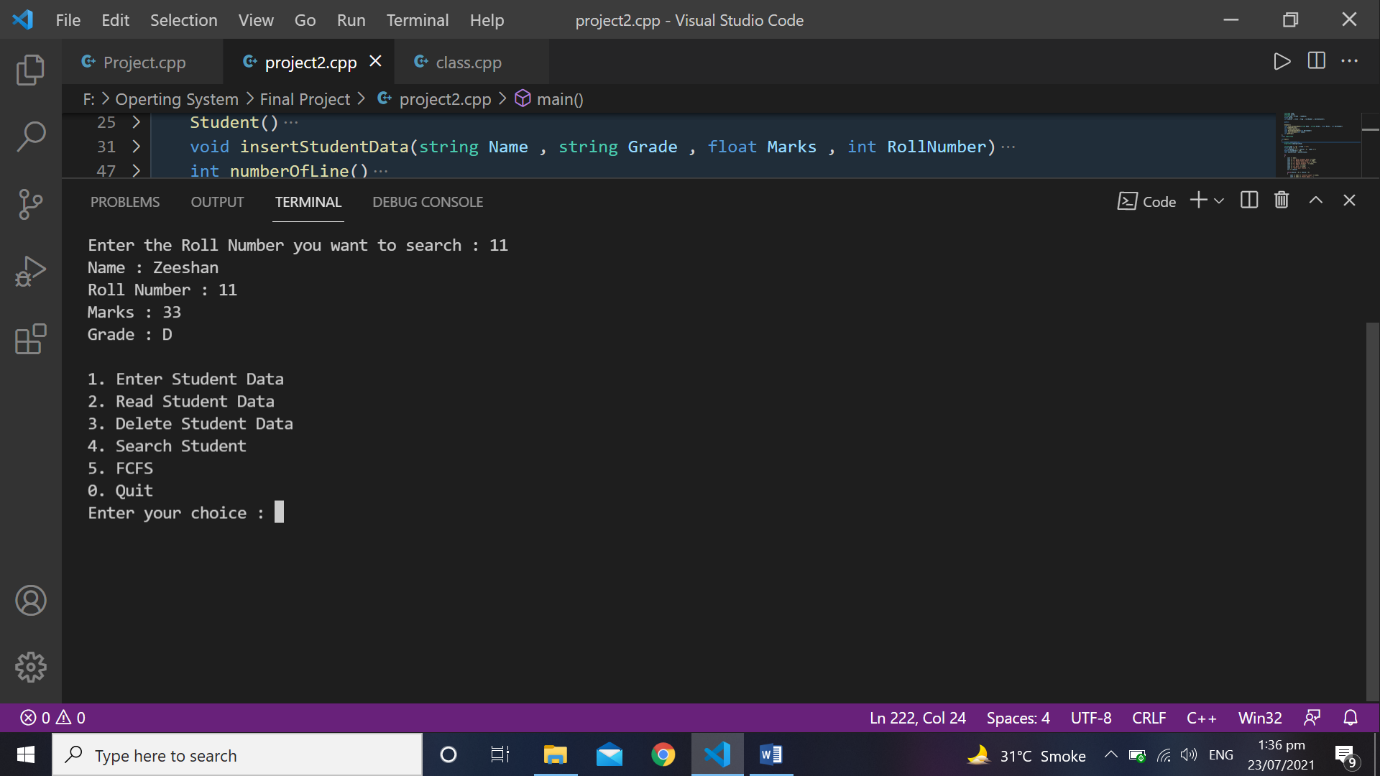
\

Figure 5.5(a)

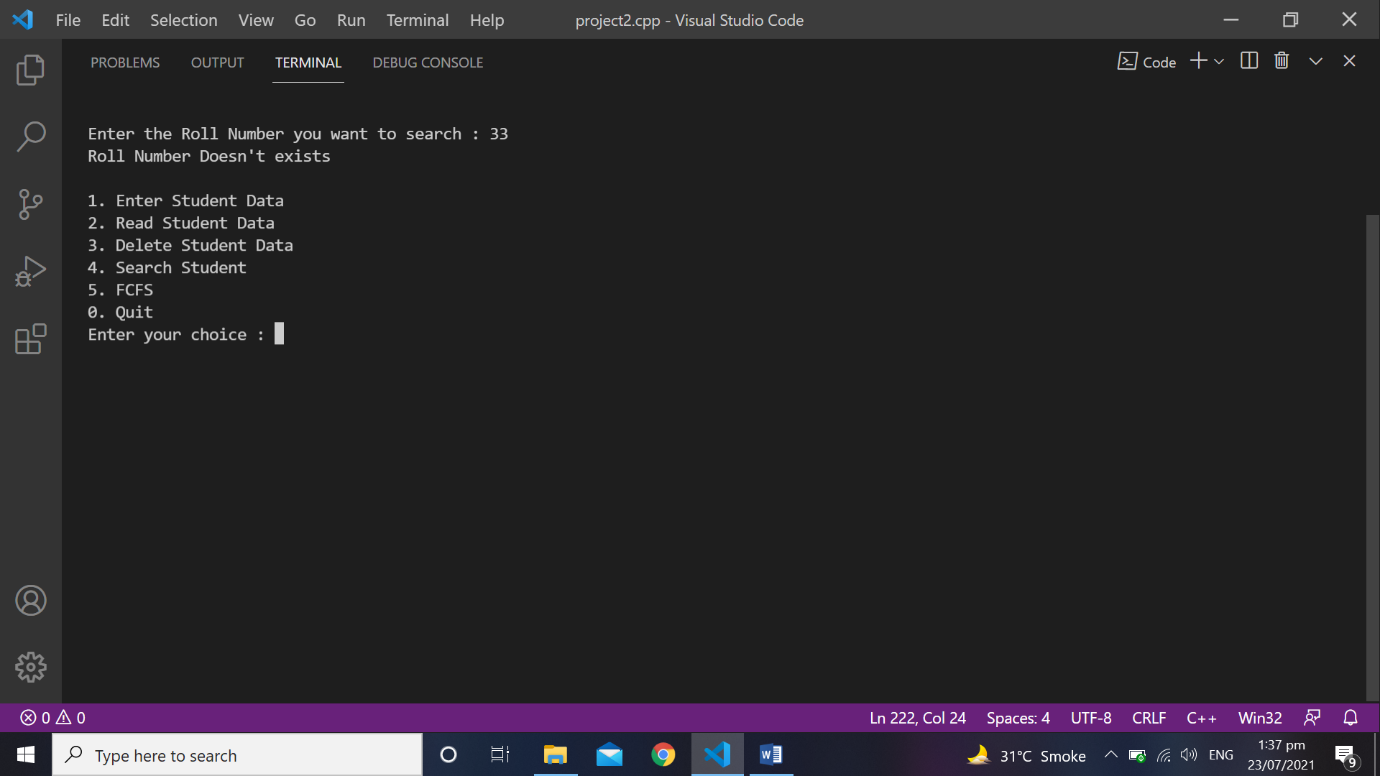


Figure 5.5(b)

## **Number of Lines**

This function calculates the total number of lines stored in the file and return that total number of lines which are further used in read and delete function.

### **Code**

int numberOfLine()

{

counter = 0;

read.open("studentData.txt");

while(getline(read , readData))

{

counter++;

}

read.close();

return counter;

}

## **Get Flag**

This function returns the value of flag.

### **Code**

int getFlag()

{

return flag;

}

# **Algorithm**

This class is used for FCFS scheduler. It contains 4 functions.

1. Get burst time and processes
2. FCFS working
3. Display
4. Get Length

## **Get burst time and processes**

This function takes the processes and time taken by the process as the parameters and store the processes in the process array and time in the burst time array.

### **Code**

void getBurstTimeAndProcess(string process , double time)

{

++length;

processes[length] = process;

burstTime[length] = time;

}

## **Get Length**

This function returns the length. Length is the total number of entries done in the array.

### **Code**

int getLength()

{

return length;

}

## **FCFS working**

In this function main working of FCFS (First Come first Serve) is done. This function take the total number of entries as the parameters.

### **Code**

void fcfsWorking(int size)

{

waitingTime[0] = 0.0;

for(int i = 1 ; i < size ; i++)

{

waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];

}

for(int i = 0 ; i < size ; i++)

{

turnAroundTime[i] = waitingTime[i] + burstTime[i];

}

}

## **Display**

This function also takes the total number of the entries as the parameters. It displays the process, burst time, waiting time and turn around time in the form of table after calculating waiting time and turn around time. It also displays average waiting and turn around time Figure 6.4.

### **Code**

void display(int size)

{

double average = 0.0 , sum = 0.0;

cout << "Process Burst Time\t Waiting Time \t Turn Around Time" << endl << endl;

for(int i = 0 ; i < size ; i++)

{

cout << processes[i] << "\t\t" << burstTime[i] << "\t\t" << waitingTime[i] << "\t\t" << turnAroundTime[i] << endl;

}

for(int i = 0 ; i < size ; i++)

{

sum = waitingTime[i] + sum;

}

average = sum / size;

cout << "Average waiting time is : " << average << endl;

average = 0.0;

sum = 0.0;

for(int i = 0 ; i < size ; i++)

{

sum = turnAroundTime[i] + sum;

}

average = sum / size;

cout << "Average turn around time is : " << average << endl;

}

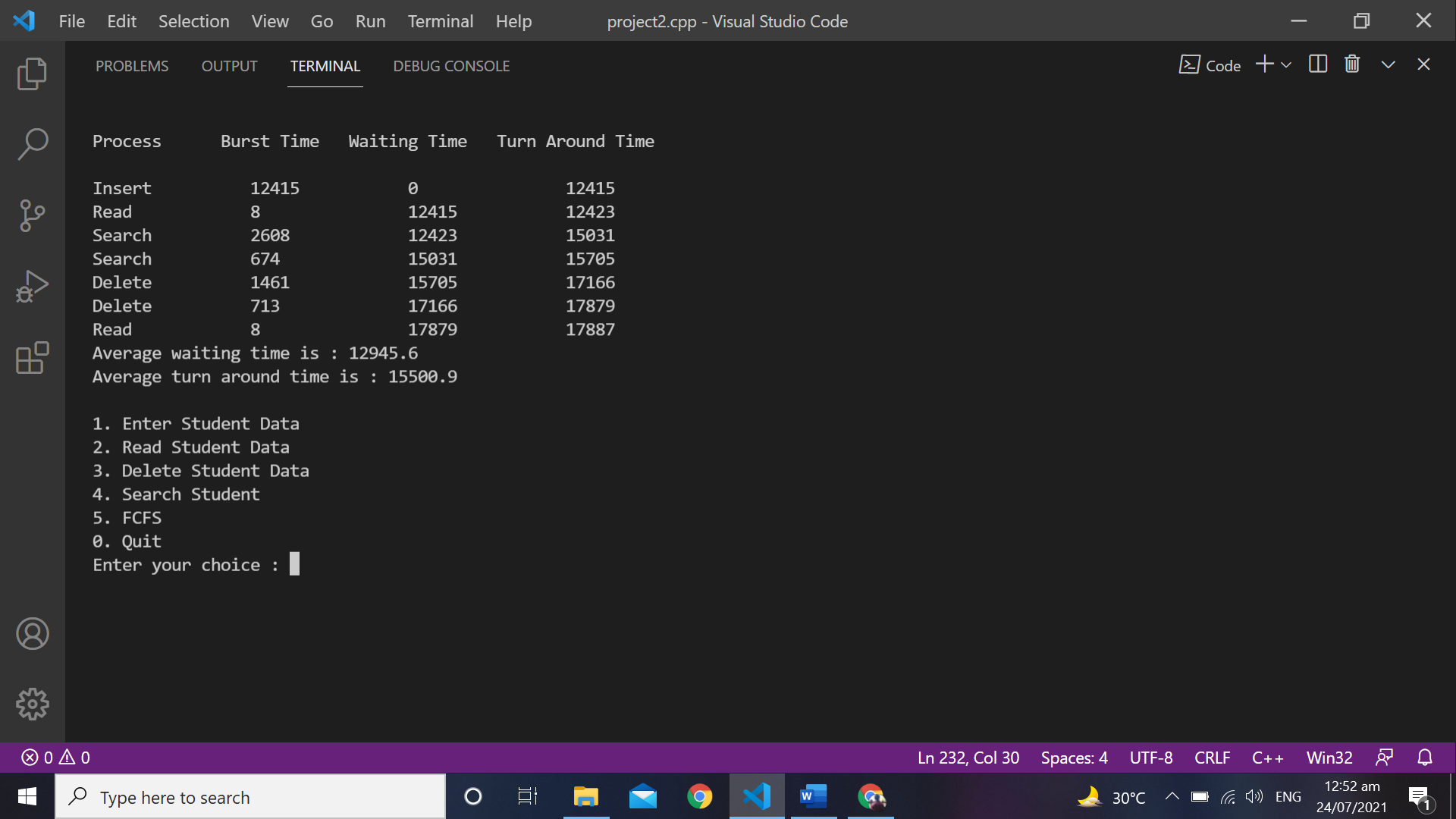


Figure 6.1

# **Int main()**

This is the main body of the program where objects of the algorithm and student class is created and their functions are called.Our main shows menu of processes between “0” to “5” Figure 7.1. If user enters the choice more than “5” or less than “0” is will displays invalid input enter again and it again takes the input from the user Figure 7.2. After taking the input it performs the operation/process. In this we used “**high\_resolution\_clock::now**”, chrono library function whose type is “**auto**” it automatically stores the starting time of the process and ending time of the process. It calculates the clock in seconds, milliseconds and nanoseconds. We used milliseconds. We select the milliseconds by using “**duration\_cast”** which is also the chrono class function. Duration cast and high-resolution clock only used for setting milliseconds and starting, ending point of the process but for calculating the total time between starting and ending time of process we use “**count**” function, this is the algorithm class function. If user presses “0” than execution of the program stops.

## **Code**

int main()

{

Student objStudent;

Algorithum objAlgorithum;

string name = "\0" , grade = "\0";

float marks = 0.0;

int rollNumber = 0 , choice = 0 , index = 0;

double storedTime = 0.0;

char nextPortion , nextPortion1;

do

{

cout << endl;

cout << "1. Enter Student Data" << endl;

cout << "2. Read Student Data" << endl;

cout << "3. Delete Student Data" << endl;

cout << "4. Search Student " << endl;

cout << "5. FCFS" << endl;

cout << "0. Quit" << endl;

cout << "Enter your choice : ";

cin >> choice;

while(choice < 0 || choice > 5)

{

cout << endl << "Invalid input" << endl;

cout << endl << "Enter Again : ";

cin >> choice;

}

system("CLS");

if(choice == 1)

{

auto starting = high\_resolution\_clock::now();

cout << "Enter name : ";

cin >> name;

cout << "Enter roll number : ";

cin >> rollNumber;

cout << "Enter marks : ";

cin >> marks;

cout << "Enter Grade : ";

cin >> grade;

objStudent.insertStudentData(name , grade , marks , rollNumber);

auto ending = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(ending - starting);

storedTime = duration.count();

objAlgorithum.getBurstTimeAndProcess("Insert" , storedTime);

}

else if(choice == 2)

{

auto starting = high\_resolution\_clock::now();

objStudent.readStudentData();

auto ending = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(ending - starting);

storedTime = duration.count();

objAlgorithum.getBurstTimeAndProcess("Read" , storedTime);

}

else if(choice == 3)

{

auto starting = high\_resolution\_clock::now();

if(objStudent.numberOfLine() == 0)

{

cout << "File is empty" << endl;

goto nextPortion;

}

cout << "Enter the Roll Number you want to delete : ";

cin >> index;

objStudent.deleteStudentData(index);

if(objStudent.getFlag() == 0)

{

cout << "Roll Number Doesn't exists" << endl;

}

nextPortion:;

auto ending = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(ending - starting);

storedTime = duration.count();

objAlgorithum.getBurstTimeAndProcess("Delete" , storedTime);

}

else if(choice == 4)

{

auto starting = high\_resolution\_clock::now();

if(objStudent.numberOfLine() == 0)

{

cout << "File is empty" << endl;

goto nextPortion1;

}

cout << "Enter the Roll Number you want to search : ";

cin >> index;

objStudent.searchStudent(index);

if(objStudent.getFlag() == 0)

{

cout << "Roll Number Doesn't exists" << endl;

}

nextPortion1:;

auto ending = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(ending - starting);

storedTime = duration.count();

objAlgorithum.getBurstTimeAndProcess("Search" , storedTime);

}

else if(choice == 5)

{

objAlgorithum.fcfsWorking(objAlgorithum.getLength() + 1);

objAlgorithum.display(objAlgorithum.getLength() + 1);

}

} while(choice != 0);

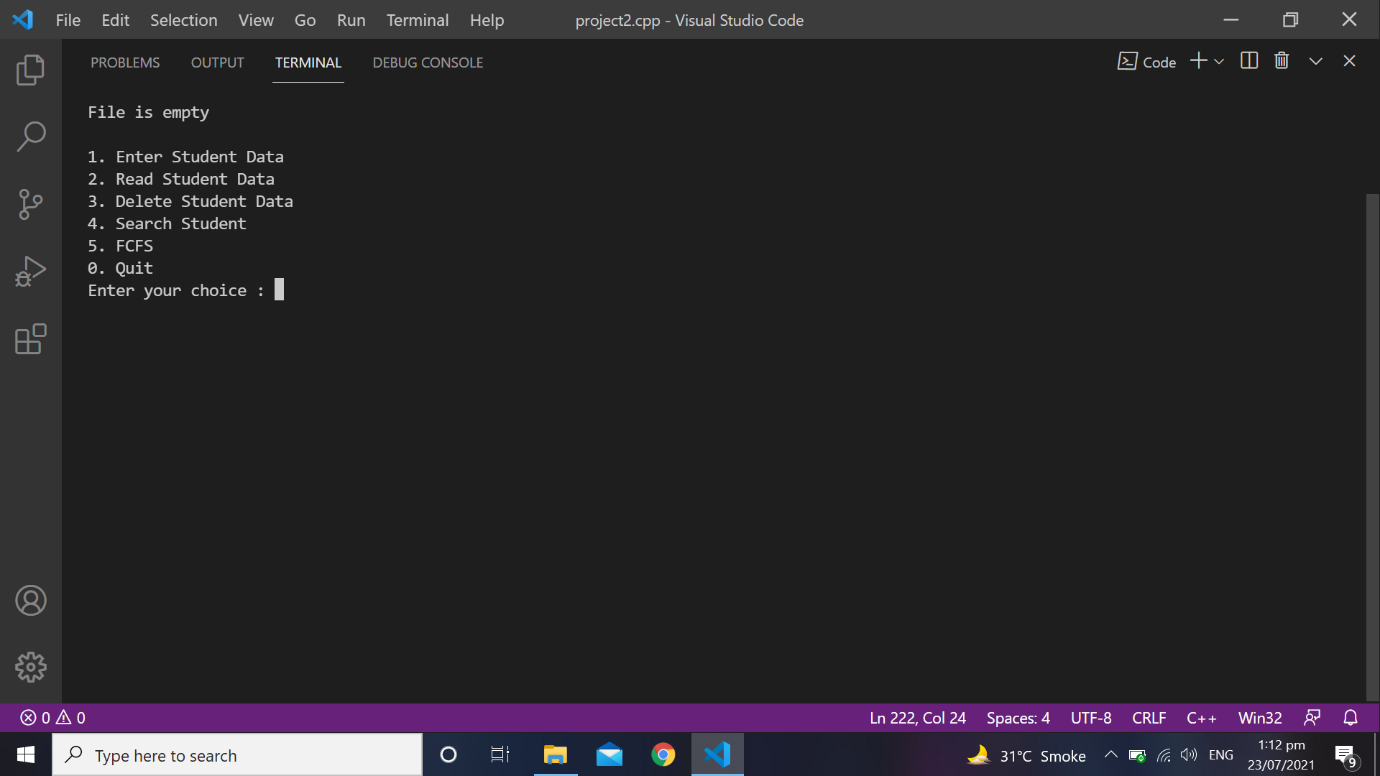
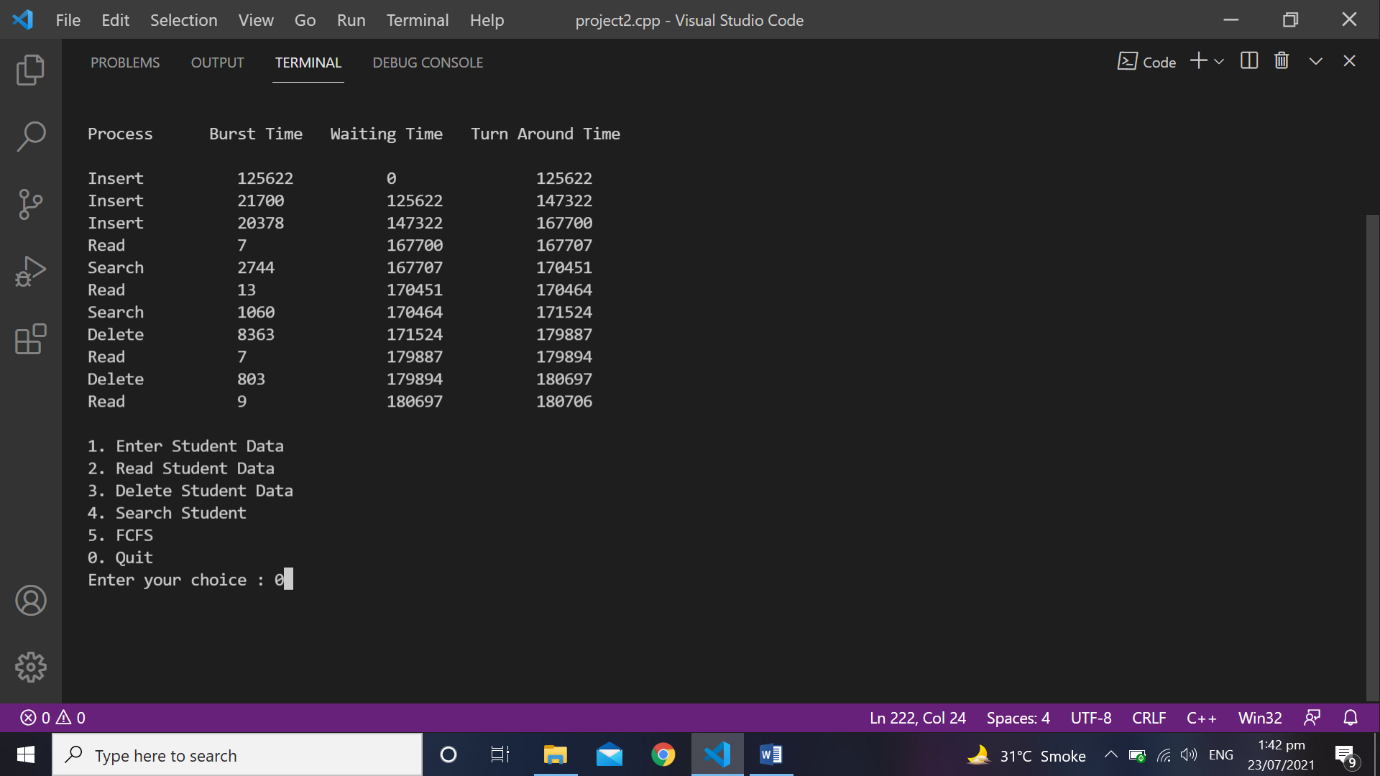


Figure 7.1



# 

# **Refrences**

* <https://www.tutorialspoint.com/operating_system/os_process_scheduling.htm>
* <https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm>
* <https://www.geeksforgeeks.org/program-for-fcfs-cpu-scheduling-set-1/>
* <https://www.programiz.com/cpp-programming/library-function/iostream#:~:text=The%20C%2B%2B%20header%20file,%2C%20cout%2C%20clog%2C%20etc>.
* <https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm>
* <https://www.geeksforgeeks.org/measure-execution-time-function-cpp/>