Computer Science Internal Assessment

Abstract :Internal Assessment of a programmatic approach/solution to solve the stress and strain of administrative workload and duties of the academic institution Oakland Group of Schools.

Group Members: Colin Campbell & Ronae Johnson Trent McKenzie

Academic Year:

2018 – 2019

Center Number:

100052

**Table of Contents**

[**Problem Definition** 1](#_Toc5904943)

[**Narrative** 2](#_Toc5904944)

[**Pseudocode** 8](#_Toc5904945)

[**The Implementation** 24](#_Toc5904946)

[**Main.c** 24](#_Toc5904947)

[**Screens.h** 27](#_Toc5904948)

[**Utils.h** 36](#_Toc5904949)

[**Test Plan** 44](#_Toc5904950)

[**Screenshots** 54](#_Toc5904951)

# **Problem Definition**

Oakland Group of Schools is located in the parish of St. Catherine Jamaica. It was founded by Pastor Oakland Whyte in the year 1984. The school, when started had only 20 students, but has rapidly grown to about 170 students over the past years. The school offers English, Math, Physical Education and Integrated Science. The school currently employs 6 teachers and 2 administrative staff, with each teacher teaching all four subjects for one class, each teacher teaches six different year groups. All student’s reports are recorded on a report card which are also stored in an unsecured file cabinet, and the school doesn’t have the budget to purchase a secured file cabinet. The teachers and administrators, when generating the students’ report card have to do it by hand, and this is described as a tedious, time-consuming and difficult activity, because of this it takes longer than should to generate one report – reports are to be generated 2 weeks before end of the term for PTA and consultation meeting, but takes longer than four weeks due to the growth of students, the date in which the report generation should start cannot be pushed back because it will conflict with the schedule for the entire staff, which is already dire and is as important. Because of this, parents start to complain about not having reports of their children in time for PTA and consultation meetings. Due to government restrictions on private schools to limit staff, and a limited budget, the school is unable to hire any more member of staff to manage these activities. A program will be created to store and generate the reports of the student using the students’ information in a text file with the file name as the name of the class, and an algorithm will be used to calculate the average for each student based on the grades that exists for the four subjects. A password and a username will be set for a class, where the administrators or a teacher, or both will have access to the login details.

# **Narrative**

**Procedure Main**

**Start**

Prompt user to add class, remove class, get class report and manage class

Get input that corresponds with the user’s actions

If user wants to add class call procedure addClass

If user wants to remove class call procedure removeClass

If user wants to get class report call procedure getClassReport

If user wants to manage class call procedure mngClass

Ask the user if they want to exit the program, if they don’t rub main again

**Stop**

**Procedure addClass**

**Start**

Prompt user for the name of the class

Create a text file with .txt extension, with the class name as the name of the class

Prompt user for their user name and password for the class

Write the username and password at the top of the file

Prompt the user to enter the amount of students they wish to enter

Prompt the user to enter the names and the average grades for the students for the subjects, integrated science, physical education, English and mathematics

Write the information from each student, line by line, with every information separated by tabs

Close the file

**Stop**

**Procedure removeClass**

**Start**

Prompt the user the name of the class

Check if the file exists

If the file exists, prompt the user their username and password for that class

Read the username and password from the top of the file

Check If the read username and password and the input username and password matches

If the corresponding variables are equal to each other call a built in method to delete the file

**Stop**

**Procedure manageClass**

**Start**

Prompt user if they would like to add student to a class or remove student from a class

Get the value that corresponds with their actions

If the user wishes to remove student from a class call sub routine removeStudent

If the user wishes to add student to a class, call sub routine addStudent

**Stop**

**Procedure addStudent**

**Start**

Prompt user for the class that they’d like to add student to

Open the file

Prompt user name and password from the user

Read the top of the file and check if the corresponding variables are equal

If the variables are equal prompt for the name of the student, they’d like to add and the grades for the four subjects

Close the reopen the file in append mode to append the student name and their grades for each subject, all separated by tab

Close the file

**Stop**

**Procedure removeStudent**

**Start**

Prompt user for the class that they’d like to add student to

Open the file

Prompt user name and password from the user

Read the top of the file and check if the corresponding variables are equal

If the variables are equal prompt for the name of the student they like to remove.

Read the file, including the username and the password at the top of the file, and the student’s information in the file

Store the student information and the teacher’s information in arrays and interim varibles

Close the file

Delete the file

Create the file with the same class name, write the teacher’s information to the newly created file, and thereafter the students’ information, skipping the student prompted to be removed using continue statement

Close the file

**Stop**

**Procedure getClassRpt**

**Start**

Prompt user for the class that they’d like to add student to

Open the file

Prompt user name and password from the user

Read the top of the file and check if the corresponding variables are equal

Read the file for students’ information into an array

Calculate the highest values for maths, English, physical education and integrates science.

Print to the screen, the students’ names, grades for each subject, and the students who score high for each subject.

**Stop**

# **Pseudocode**

|  |
| --- |
| **Start Algorithm**  exit 🡨 0 |
| option 🡨 0 |
|  |
| while ( exit 🡨 0) do |
|  |
| string decision 🡨 0 |
| option 🡨 menu() |
| DoCase (option) |
|  |
| case 1 : addClass() |
| break |
| case 2 : removeClass() |
| break |
| case 3 : getClassRpt() |
| break |
| case 4 : mngClass() |
| break |
| case 5 : break |
| EndCase |
|  |
| Write("Would you like to terminate the program ? y/n ") |
| fflush(stdin) |
| Read (decision) |
|  |
| If ( decision 🡨 “y”) |
| exit 🡨 1 |
| else If (decision 🡨 “n”) |
| exit 🡨 0 |
| else |
| EndIf |
|  |
| Write("Sorry the input is not regonised, try using lower case, not upper case letters") |
| Write("Would you like to terminate the program ? y/n ") |
| fflush(stdin) |
| Read (decision) |
|  |
| If ( decision 🡨 “y”) |
| exit 🡨 1 |
| else If (decision 🡨 “n”) |
| exit 🡨 0 |
| EndIf |
| Write (NEWLINE) |
| EndWhile |
| **Stop Algorithm** |

|  |
| --- |
| **Start Algorithm : int menu()** |
| option 🡨 0 |
|  |
| Write("Please chose the number that corresponds with the actions you will like to commit :") |
| Write(NEWLINE) |
| Write("1) Add a class.") |
| Write(NEWLINE) |
| Write("2) Remove a class") |
| Write(NEWLINE) |
| Write("3) Get a class report") |
| Write(NEWLINE) |
| Write("4) Manage class") |
| Write(NEWLINE) |
| Write("5) Undecided") |
| Write(NEWLINE) |
|  |
|  |
| Write("What do wish to do : ") |
| Read(option) |
| return option |
| **Stop Algorithm** |
|  |
| **Start Algorithm: void addClass()** |
|  |
| string cName |
| string tName |
| string tPass |
|  |
| Write(NEWLINE) |
| Write("|\t|\t|\t|\t Add Class \t|\t|\t|\t|") |
| Write(NEWLINE) |
|  |
| Write("Please enter the class's name : ") |
|  |
| Read(cName) |
|  |
| Write("Please enter username : ") |
|  |
| Read(tName) |
|  |
| Write ("Please enter password for the class : ") |
|  |
| Read(tPass) |
|  |
| createClass(cName,tName,tPass) |
| **Stop Algorithm** |
|  |
| **Start Algorithm: void removeClass()** |
|  |
| string className |
| string tUsername |
| string tPassword |
|  |
| Write ("Please enter the name of the class you'd like to delete : ") |
|  |
| Read(className) |
|  |
| strcat(className,".txt") |
|  |
|  |
| OpenFile “className” for Read |
| If (!fp) then |
|  |
| Write("Class not found",className) |
| CloseFile “className” |
|  |
| EndIF |
|  |
| Write("Please enter username : ") |
|  |
| Read(tUsername) |
| Write("Please enter password for class : ") |
|  |
| Read(tPassword) |
| If (fp) then |
|  |
| string readUsername |
| string readPassword |
| Read(readUsername,readPassword) from “className” file |
| Close “className” file |
| If (strcmp(readPassword,tPassword) 🡨 0 and |
| strcmp(readUsername,tUsername) 🡨 0) then |
|  |
| status 🡨 remove(className) |
| Write ("Status :",status) |
| else |
| Write ("User name and/or password is incorrect") |
| else |
|  |
| Write ("File not found",className) |
| EndIf  EndIf  EndIf  EndIf  **Stop Algorithm** |
|  |
|  |
| **Start Algorithm: void getClassRpt()** |
|  |
| string tUsername |
| string tPassword |
| string className |
|  |
| Write (NEWLINE) |
| Write ("|\t|\t|\t|\t Class Report \t|\t|\t|\t|") |
| Write (NEWLINE) |
| Write (NEWLINE) |
|  |
| Write ("Please enter class you'd like to get report from : ") |
|  |
| Read(className) |
|  |
| strcat(className,".txt") |
| OpenFile className for Read |
|  |
| Write ("Please enter usename : ") |
|  |
| Read(tUsername) |
|  |
| Write ("Please enter password : ") |
|  |
| Read(tPassword) |
|  |
| bool isLogged 🡨 loginTeacher(tUsername,tPassword,fp) |
| struct Student students[100] |
| If (isLogged) then |
| x 🡨 0 |
| i 🡨 0 |
|  |
| string heighestAverageStudent |
| heighestAverage 🡨 0 |
|  |
| string heighestEngStudent |
| heighestEng 🡨 0 |
|  |
| string heighestPhysEdStudent |
| heighestPhyEd 🡨 0 |
|  |
| string heighestMathStudent |
| heighestMath 🡨 0 |
|  |
| string heighestInteSciStudent |
| heighestInteSci 🡨 0 |
|  |
| While (x ! 🡨 EOF) |
| struct Student student |
| x🡨Read(student.fullName,student.mathGrade,student.engGrade,student.phyEdGrade,student.inteSciGrade) from className file |
|  |
|  |
| strcpy(students[i].fullName,student.fullName) |
| students[i].mathGrade 🡨 student.mathGrade |
| students[i].engGrade 🡨 student.engGrade |
| students[i].phyEdGrade 🡨 student.phyEdGrade |
| students[i].inteSciGrade 🡨 student.inteSciGrade |
|  |
|  |
| If (students[i].mathGrade > heighestMath) then |
|  |
| heighestMath 🡨 students[i].mathGrade |
| strcpy(heighestMathStudent,students[i].fullName) |
| EndIf |
|  |
| If (students[i].engGrade > heighestEng) then |
|  |
| heighestEng 🡨 students[i].engGrade |
| strcpy(heighestEngStudent,students[i].fullName) |
| EndIf |
|  |
| If (students[i].phyEdGrade > heighestPhyEd) then |
|  |
| heighestPhyEd 🡨 students[i].phyEdGrade |
| strcpy(heighestPhysEdStudent,students[i].fullName) |
| EndIf |
|  |
| If (students[i].inteSciGrade > heighestInteSci) then |
|  |
| heighestInteSci 🡨 students[i].inteSciGrade |
| strcpy(heighestInteSciStudent,students[i].fullName) |
| EndIf |
| i++ |
| EndWhile |
|  |
| i 🡨 i-1 |
| j 🡨 0 |
| Write("Student Name/Math Average Grade/English Average Grade/Integrated Science Average Grade/Physical Education Average Grade/Overall Average) |
| While ( j < i) |
|  |
| averageGrade 🡨 (students[j].mathGrade + students[j].engGrade + students[j].inteSciGrade + students[j].phyEdGrade)/4  EndWhile |
| Write(students[j].fullName,students[j].mathGrade,students[j].engGrade,students[j].inteSciGrade,students[j].phyEdGrade,averageGrade) |
| j++ |
| EndWhile |
|  |
| Write("Heighest average in mathematics ",heighestMath,heighestMathStudent) |
| Write("Heighest average in english",heighestEng,heighestEngStudent) |
| Write("Heighest average in integrated science  ",heighestInteSci,heighestInteSciStudent) |
| Write("Heighest average in physical education ",heighestPhyEd,heighestPhysEdStudent) |
|  |
| EndIf |
|  |
| **Stop Algorithm** |
|  |
| **Start Algorithm: void mngClass()** |
|  |
| option 🡨 0 |
|  |
| Write (NEWLINE) |
| Write ("|\t|\t|\t|\t Manage Class \t|\t|\t|\t|") |
| Write (NEWLINE) |
| Write (NEWLINE) |
|  |
| Write ("Please chose the number that corresponds with the actions you will like to commit :") |
| Write (NEWLINE) |
| Write (NEWLINE) |
|  |
| Write ("1) Add Student to a class.") |
| Write (NEWLINE) |
|  |
| Write ("2) Remove Student from a class.") |
| Write (NEWLINE) |
|  |
| Write ("What would like to do : ") |
| Read(option) |
|  |
| If (option 🡨 1) then |
| addStudents\_Abstract() |
| else If(option 🡨 2) |
| removeStudent() |
| else |
|  |
| Write ("The value you entered is not recognised") |
| Write (NEWLINE) |
| return |
| EndIf |
| |  | | --- | |  | | **Start Algorithm: void createClass( string cName, string tName , string tPass )** | |  | | FILE \*ptr | |  | | string \*fileExtension 🡨 ".txt" | |  | | strcat(cName,fileExtension) | | OpenFile cName for Write | |  | |  | | If (ptr) then | |  | | Write(tName,tPass) to “cName” file | | addStudents(ptr) | | Write("Class Created successfully") | | CloseFile “cName” | | else | |  | | Write (NEWLINE) | | perror("Error") | | Write (NEWLINE) | | exit(1) | | EndIf | | **Stop Algorithm** | |  | | **Start Algorithm: bool loginTeacher( string tName , string tPass ,FILE \*fp)** | |  | |  | | If (fp) then | |  | | string readUsername | | string readPassword | | Read(readUsername,readPassword) from “className” file | |  | | If ( readPassword = tPass AND | | ReadUsername = tName ) then | | return true | | else | |  | | Write ("User name and/or password is incorrect") | | return false | |  | | else | |  | | return false | | EndIf | | EndIf  EndIf | | EndIf  **Stop Algorithm** | |  | | **Start Algorithm: void addStudents(FILE \*fp)** | |  | | size 🡨 0 | |  | | Write ("Please enter the amount of students that are in the class :") | | Read(size) | | struct Student students[size] | | For (i 🡨 0 i < size) do | |  | | x 🡨 i + 1 | | Write ("Please enter student full name : ",x) | |  | | Read(students[i].fullName) | |  | | Write ("Please enter student Mathematics average grade : ",x) | |  | | Read (students[i].mathGrade) | |  | | Write ("Please enter student English average grade : ",x) | |  | | Read (students[i].engGrade) | |  | | Write ("Please enter student Physical Education average grade : ",x) | |  | | Read (students[i].phyEdGrade) | |  | | Write ("Please enter student Integrated Science average grade : ",x) | |  | | Read (students[i].inteSciGrade) | | EndFor | |  | | For ( i 🡨 0 i < size) do | | Write(fp,students[i].fullName,students[i].mathGrade,students[i].engGrade,students[i].phyEdGrade,students[i].inteSciGrade) to “className” file | | EndFor | | **Stop Algorithm** | |  | | **Start Algorithm: void addStudents\_Abstract()** | |  | | string tUsername | | string tPass | | string className | |  | | Write ("Please enter the class you'd like to add the student to : ") | |  | | Read(className) | |  | | strcat(className,".txt") | | OpenFile “className” for Read | |  | | Write ("Please enter username : ") | |  | | Read(tUsername) | |  | | Write ("Please enter password : ") | |  | | Read(tPass) | |  | | bool isLogged 🡨 loginTeacher(tUsername,tPass,fp) | |  | | OpenFile “className” for Appending | |  | | If (isLogged) then   |  |  | | --- | --- | |  | Write("Please enter the amount of students you would like to add.") | |  | numStud 🡨 0 | |  | fflush(stdin) | |  | Read (numStud) | |  |  | |  | struct Student students[numStud] | | |  | | For (i 🡨 0 i < numStud) do | | a 🡨 i + 1  Write ("Please enter student full name : ") | |  | | Read(student.fullName) | |  | | Write ("Please enter student Mathematics average grade : ") | |  | | Read (student.mathGrade) | |  | | Write ("Please enter student English average grade : ") | |  | | Read (student.engGrade) | |  | | Write ("Please enter student Physical Education average grade : ") | |  | | Read (student.phyEdGrade) | |  | | Write ("Please enter student Integrated Science average grade : ") | |  | | Read (student.inteSciGrade) | | WriteFile(fp,student.fullName,student.mathGrade,student.engGrade,student.phyEdGrade,student.inteSciGrade) | | Write ("Student added successfully")    EndFor  CloseFile “className” | |  | |  | | EndIf | | **Stop Algorithm** | |  | | **Start Algorithm: void removeStudent()** | |  | | string tUsername | | string tPass | | string className | | string studentName | |  | | Write ("Please enter the class you'd like to add the student to : ") | |  | | Read(className) | |  | | strcat(className,".txt"); // add .txt to the file extension | | OpenFile “className” Read | |  | | Write ("Please enter username : ") | |  | | Read(tUsername) | |  | | Write ("Please enter password : ") | |  | | Read(tPass) | |  | | Write ("Please enter student that you'd like to remove : ") | |  | | Read(studentName) | |  | | struct Student students[100] | |  | | bool isLogged 🡨 loginTeacher(tUsername,tPass,fp) | | If (isLogged) then | |  | | x 🡨 1 | | i 🡨 0 | |  | | struct Student student | |  | | while (x ! 🡨 EOF) | | x🡨Write(fp,student.fullName,&student.mathGrade,&student.engGrade,&student.phyEdGrade,&student.inteSciGrade) to “className” file | | Write ("Name : i 🡨", student.fullName, i) | |  | | strcpy(students[i].fullName,student.fullName) | | students[i].mathGrade 🡨 student.mathGrade | | students[i].engGrade 🡨 student.engGrade | | students[i].phyEdGrade 🡨 student.phyEdGrade | | students[i].inteSciGrade 🡨 student.inteSciGrade | | i++ | | EndWhile | |  | |  | | CloseFile “className” | | remove(className) | | j 🡨 i | |  | | OpenFile “className” for Write | | Write (tUsername,tPass) to “cName” file | |  | | length 🡨 sizeof(students) / sizeof(students[0]) | | Write (" j :",j) | |  | | For ( i 🡨 0; i < length) do | |  | | If ( strcmp(studentName,students[i].fullName) 🡨 0) | | continue | | else If ( i 🡨 j) | | break | | Write(ptr,,students[i].fullName,students[i].mathGrade,students[i].engGrade,students[i].phyEdGrade,students[i].inteSciGrade) to cName file | | EndFor | |  | | EndIf | | **Stop Algorithm** | |

# **The Implementation**

## **Main.c**

#include <stdio.h>

#include <stdlib.h>

#include "screens.h"

#define NEWLINE "\n"

// @author Colin A. Campbell

// Entry point for program is here

int main()

{

int exit = 0; // sentinel value for the while loop

int option = 0; // value returned to from menu

printf("---------- Oakland Group of Schools Information System ----------\n\n");

while ( exit == 0)

{

char decision = 0;

option = menu();

switch (option)

{

case 1 : addClass();

break;

case 2 : removeClass();

break;

case 3 : getClassRpt();

break;

case 4 : mngClass();

break;

case 5: break;

}

printf("\nWould you like to terminate the program ? y/n ");

fflush(stdin); // flush the buffer so it can store other values

scanf("%c",&decision);

if ( decision == 'y')

exit = 1; // 1 means true

else if (decision == 'n')

exit = 0; // 0 mean false

else

{

// Ask the user a second time, if the first input for decision is not recognised

printf("Sorry the input is not regonised, try using lower case, not upper case letters");

printf("Would you like to terminate the program ? y/n ");

fflush(stdin);

scanf("%c",&decision);

if ( decision == 'y')

exit = 1; // 1 means true

else if (decision == 'n')

exit = 0;

}

printf(NEWLINE);

}

}

## **Screens.h**

#include "utils.h"

#define NEWLINE "\n"

extern FILE \*ptr;

int menu()

{

int option = 0;

printf("Please chose the number that corresponds with the actions you will like to commit :");

printf(NEWLINE);

printf("1) Add a class.");

printf(NEWLINE);

printf("2) Remove a class");

printf(NEWLINE);

printf("3) Get a class report");

printf(NEWLINE);

printf("4) Manage class");

printf(NEWLINE);

printf("5) Undecided");

printf(NEWLINE);

flushBuffer();// flush out the buffer so it wont overflow fresh values

printf("What do wish to do : ");

scanf("%d",&option);

return option;

}

void addClass()

{

char cName[40]; // class name

char tName[40]; // teacher's name

char tPass[40]; // teacher's password

printf(NEWLINE);

printf("|\t|\t|\t|\t Add Class \t|\t|\t|\t|");

printf(NEWLINE);

printf("Please enter the class's name : ");

flushBuffer();

gets(cName);

printf("Please enter username : ");

flushBuffer();

gets(tName);

printf("Please enter password for the class : ");

flushBuffer();

gets(tPass);

createClass(cName,tName,tPass);

}

void removeClass()

{

char className[40];

char tUsername[40];

char tPassword[40];

// TODO finish up this method

printf("Please enter the name of the class you'd like to delete : ");

flushBuffer();

gets(className);

strcat(className,".txt");

// open file to read line by line

FILE \*fp; // fp : file pointer

fp = fopen(className,"r");

if (!fp)

{

printf("Class %s, not found",className);

fclose(fp);

return; // exit the function if fp is null

}

printf("Please enter username : ");

flushBuffer();

gets(tUsername);

printf("Please enter password for class : ");

flushBuffer();

gets(tPassword);

if (fp)

{

char readUsername[40];

char readPassword[40];

fscanf(fp,"%s\t%s",readUsername,readPassword);

fclose(fp); // close the pointer to the file so, the file can be deleted safely

if (strcmp(readPassword,tPassword) == 0 && // strcmp, or string compare, returns 0 if both strings are equal to each other

strcmp(readUsername,tUsername) == 0)

{

int status = remove(className);

printf("Status : %d",status);

} else

printf("User name and/or password is incorrect");

} else

{

printf("File %s, not found\n",className);

}

}

void getClassRpt()

{

char tUsername[40];

char tPassword[40];

char className[40];

printf(NEWLINE);

printf("|\t|\t|\t|\t Class Report \t|\t|\t|\t|");

printf(NEWLINE);

printf(NEWLINE);

printf("Please enter class you'd like to get report from : ");

flushBuffer();

gets(className);

strcat(className,".txt");

FILE \*fp = fopen(className,"r");

printf("Please enter usename : ");

flushBuffer();

gets(tUsername);

printf("Please enter password : ");

flushBuffer();

gets(tPassword);

bool isLogged = loginTeacher(tUsername,tPassword,fp);

struct Student students[100];

if (isLogged)

{

// read scan student information to the file

int x = 0;

int i = 0;

char heighestAverageStudent[40];

int heighestAverage = 0;

char heighestEngStudent[40];

int heighestEng = 0;

char heighestPhysEdStudent[40];

int heighestPhyEd = 0;

char heighestMathStudent[40];

int heighestMath = 0;

char heighestInteSciStudent[40];

int heighestInteSci = 0;

while (x != EOF)

{

struct Student student;

x = fscanf(fp,"%s\t%d\t%d\t%d\t%d\n",student.fullName,&student.mathGrade,&student.engGrade,&student.phyEdGrade,&student.inteSciGrade);

//printf("Name : %s, i = %d\n",student.fullName,i);

strcpy(students[i].fullName,student.fullName);

students[i].mathGrade = student.mathGrade;

students[i].engGrade = student.engGrade;

students[i].phyEdGrade = student.phyEdGrade;

students[i].inteSciGrade = student.inteSciGrade;

// calculate the heighest scores

if (students[i].mathGrade > heighestMath)

{

heighestMath = students[i].mathGrade;

strcpy(heighestMathStudent,students[i].fullName);

}

if (students[i].engGrade > heighestEng)

{

heighestEng = students[i].engGrade;

strcpy(heighestEngStudent,students[i].fullName);

}

if (students[i].phyEdGrade > heighestPhyEd)

{

heighestPhyEd = students[i].phyEdGrade;

strcpy(heighestPhysEdStudent,students[i].fullName);

}

if (students[i].inteSciGrade > heighestInteSci)

{

heighestInteSci = students[i].inteSciGrade;

strcpy(heighestInteSciStudent,students[i].fullName);

}

i++;

}

i = i-1;

int j = 0;

printf("\nStudent Name/Math Average Grade/English Average Grade/Integrated Science Average Grade/Physical Education Average Grade/Overall Average\n");

while ( j < i)

{

int averageGrade = (students[j].mathGrade + students[j].engGrade + students[j].inteSciGrade + students[j].phyEdGrade)/4;

printf("%s\t %d \t %d \t %d\t %d\t %d\n",students[j].fullName,students[j].mathGrade,students[j].engGrade,students[j].inteSciGrade,students[j].phyEdGrade,averageGrade);

j++;

}

printf("\nHeighest average in mathematics %d, %s\n",heighestMath,heighestMathStudent);

printf("Heighest average in english %d, %s\n",heighestEng,heighestEngStudent);

printf("Heighest average in integrated science %d, %s\n",heighestInteSci,heighestInteSciStudent);

printf("Heighest average in physical education %d, %s\n",heighestPhyEd,heighestPhysEdStudent);

}

}

void mngClass()

{

int option = 0;

printf(NEWLINE);

printf("|\t|\t|\t|\t Manage Class \t|\t|\t|\t|");

printf(NEWLINE);

printf(NEWLINE);

printf("Please chose the number that corresponds with the actions you will like to commit :");

printf(NEWLINE);

printf(NEWLINE);

printf("1) Add Students to a class.");

printf(NEWLINE);

printf("2) Remove Student from a class.");

printf(NEWLINE);

printf("\nWhat would like to do : ");

scanf("%d",&option);

if (option == 1)

addStudents\_Abstract(); // subscreen

else if(option == 2)

removeStudent(); // subscreen

else

{

printf("The value you entered is not recognised");

printf(NEWLINE);

return;

}

}

## **Utils.h**

void flushBuffer()

{

fflush(stdin);

}

// method used to create class file

// return the pointer so it can be used by other methods

void createClass(char cName[60], char tName[40],char tPass[40])

{

FILE \*ptr;

char \*fileExtension = ".txt";

strcat(cName,fileExtension);

ptr = fopen(cName,"w");

// Error handling

if (ptr)

{

fprintf(ptr,"%s\t%s\n",tName,tPass); // put the user name and password at top of the file

//addStudents(ptr);

printf("\nClass Created successfully");

fclose(ptr);

} else

{

printf(NEWLINE);

perror("Error");

printf(NEWLINE);

exit(1);

}

}

bool loginTeacher(char tName[40],char tPass[40],FILE \*fp)

{

// copied from removeClass function

if (fp)

{

char readUsername[40];

char readPassword[40];

fscanf(fp,"%s\t%s\n",readUsername,readPassword);

//fclose(fp); // close the pointer to the file so, the file can be deleted safely

if (strcmp(readPassword,tPass) == 0 && // strcmp, or string compare, returns 0 if both strings are equal to each other

strcmp(readUsername,tName) == 0)

{

return true;

} else

{

printf("User name and/or password is incorrect");

return false;

}

} else

{

return false;

}

}

// Subscreen

void addStudents(FILE \*fp)

{

int size = 0;

flushBuffer();

printf("Please enter the amount of students that are in the class :");

scanf("%d",&size);

struct Student students[size];

// Read students' information into an array

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

{

int x = i + 1;

printf("\nPlease enter student %d full name : ",x);

flushBuffer();

gets(students[i].fullName);

printf("Please enter student %d Mathematics average grade : ",x);

flushBuffer();

scanf("%d",&students[i].mathGrade);

printf("Please enter student %d English average grade : ",x);

flushBuffer();

scanf("%d",&students[i].engGrade);

printf("Please enter student %d Physical Education average grade : ",x);

flushBuffer();

scanf("%d",&students[i].phyEdGrade);

printf("Please enter student %d Integrated Science average grade : ",x);

flushBuffer();

scanf("%d",&students[i].inteSciGrade);

}

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

{

fprintf(fp,"%s\t%d\t%d\t%d\t%d\n",students[i].fullName,students[i].mathGrade,students[i].engGrade,students[i].phyEdGrade,students[i].inteSciGrade);

}

}

// difference between addStudents and addStudents\_Abstract is that the latter does not takes in

// a pointer to a file, it already asks for one

void addStudents\_Abstract()

{

char tUsername[40];

char tPass[40];

char className[40];

printf("\nPlease enter the class you'd like to add the student to : ");

flushBuffer();

gets(className);

strcat(className,".txt"); // add .txt to the file extension

FILE \*fp = fopen(className,"r");

printf("Please enter username : ");

flushBuffer();

gets(tUsername);

printf("Please enter password : ");

flushBuffer();

gets(tPass);

bool isLogged = loginTeacher(tUsername,tPass,fp);

fp = fopen(className,"a"); // open the file again, it was closed in loginTeacher

if (isLogged)

{

printf("Please enter the amount of students you would like to add.");

int numStud = 0; // number students

fflush(stdin);

scanf("%d",&numStud);

struct Student students[numStud];

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

{

int a = i + 1;

printf("\nPlease enter student %d full name : ",a);

flushBuffer();

gets(students[i].fullName);

printf("Please enter student Mathematics average grade : ");

flushBuffer();

scanf("%d",&students[i].mathGrade);

printf("Please enter student English average grade : ");

flushBuffer();

scanf("%d",&students[i].engGrade);

printf("Please enter student Physical Education average grade : ");

flushBuffer();

scanf("%d",&students[i].phyEdGrade);

printf("Please enter student Integrated Science average grade : ");

flushBuffer();

scanf("%d",&students[i].inteSciGrade);

fprintf(fp,"%s\t%d\t%d\t%d\t%d\n",students[i].fullName,students[i].mathGrade,students[i].engGrade,students[i].phyEdGrade,students[i].inteSciGrade);

printf("Student added successfully\n");

}

fclose(fp);

}

}

void removeStudent()

{

char tUsername[40];

char tPass[40];

char className[40];

char studentName[40];

printf("\nPlease enter the class you'd like to add the student to : ");

flushBuffer();

gets(className);

strcat(className,".txt"); // add .txt to the file extension

FILE \*fp = fopen(className,"r");

printf("Please enter username : ");

flushBuffer();

gets(tUsername);

printf("Please enter password : ");

flushBuffer();

gets(tPass);

printf("Please enter student that you'd like to remove : ");

flushBuffer();

gets(studentName);

struct Student students[100];

bool isLogged = loginTeacher(tUsername,tPass,fp);

if (isLogged)

{

int x = 1;

int i = 0;

struct Student student;

while (x != EOF)

{

x = fscanf(fp,"%s\t%d\t%d\t%d\t%d\n",student.fullName,&student.mathGrade,&student.engGrade,&student.phyEdGrade,&student.inteSciGrade);

printf("Name : %s, i = %d\n",student.fullName,i);

strcpy(students[i].fullName,student.fullName);

students[i].mathGrade = student.mathGrade;

students[i].engGrade = student.engGrade;

students[i].phyEdGrade = student.phyEdGrade;

students[i].inteSciGrade = student.inteSciGrade;

i++;

}

// close the pointer and remove the class

fclose(fp);

remove(className);

int j = i;

FILE \*ptr = fopen(className,"w");

fprintf(ptr,"%s\t%s\n",tUsername,tPass);

int length = sizeof(students) / sizeof(students[0]);

printf("\n j : %d",j);

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

{

if ( strcmp(studentName,students[i].fullName) == 0)

continue;

else if ( i == j)

break;

fprintf(ptr,"%s\t%d\t%d\t%d\t%d\n",students[i].fullName,students[i].mathGrade,students[i].engGrade,students[i].phyEdGrade,students[i].inteSciGrade);

}

}

}

# **Test Plan**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | Pass | Fail |
| option | integer | 1 | Allow the program to store and execute the user option from the menu to add a class | Pass |  |
| cName | String | ? | To allow program to store the name of the form teacher class | Pass |  |
| tName | String | user | To allow program to store the name of the class teacher | Pass |  |
| tPass | String | pass | To grant the user full access to the program | Pass |  |
| Program is ran again |  |  |  |  |  |
| readUsername | | | | | | String | users@3 | To ensure the right username is enter to allow the user access | Pass |  |
| readPassword | String | Password112 | To ensure the right password is enter to allow the user access | Pass |  |
| size | Integer | 3 | Allow the program to repeat the program a set amount of time | Pass |  |
| x | integer | 1 | Allow the program to count the number of students for each iteration | Pass |  |
| Students[i].fullName | String | Jan | Allow the program to store all students full name to a file |  |  |
| Students[i].mathGrade | *String* | *100* | Allow the program to store all math students grades to a file |  |  |
| Students[i].engGrade | String | *29* | Allow the program to store all English students grades name to a file |  |  |
| Students[i].phyEdGrade | String | *42* | Allow the program to store all Physical Education students grades to a file |  |  |
| Students[i].inteSciGrade | String | *39* | Allow the program to store all Integrated Science Students grades to a file |  |  |
| x | integer | 2 | Allow the program to count the number of students for each iteration |  |  |
| Students[i].fullName | String | Wan | Allow the program to store all students full name to a file |  |  |
| Students[i].mathGrade | *String* | 23 | Allow the program to store all math students grades to a file |  |  |
| Students[i].engGrade | String | 49 | Allow the program to store all English students grades name to a file |  |  |
| Students[i].phyEdGrade | String | 11 | Allow the program to store all Physical Education students grades to a file |  |  |
| Students[i].inteSciGrade | String | 29 | Allow the program to store all Integrated Science Students grades to a file |  |  |
| x | integer | 3 | Allow the program to count the number of students for each iteration |  |  |
| Students[i].fullName | String | Han | Allow the program to store all students full name to a file |  |  |
| Students[i].mathGrade | *String* | 83 | Allow the program to store all math students grades to a file |  |  |
| Students[i].engGrade | String | 34 | Allow the program to store all English students grades name to a file |  |  |
| Students[i].phyEdGrade | String | 98 | Allow the program to store all Physical Education students grades to a file |  |  |
| Students[i].inteSciGrade | String | 73 | Allow the program to store all Integrated Science Students grades to a file |  |  |
| Program ran again |  |  |  |  |  |
| option | | | | | | integer | 4 | Allow the program to store and execute the user option from the menu to manage a class | Pass |  |
| option | integer | 1 | Allow the program to store and execute the user option from the menu to add a student under class management | Pass |  |
| tUsername | String | user | To allow program to store the name of the class teacher | Pass |  |
| tPassword | String | pass | To grant the user full access to the program | Pass |  |
| className | String |  | To allow program to store the name of the form teacher class | Pass |  |
| Student[i].fullName |  | Pan | Allow program store full name of a student | Pass |  |
| Student[i].mathGrade |  | 45 | Allow the program to store the math grade of a student | Pass |  |
| Student[i].engGrade |  | 67 | Allow the program to store the English grade of a student | Pass |  |
| Student[i].inteSciGrade |  | 87 | Allow the program to store the Integrated Science grade of the student | Pass |  |
| Student[i].phyEdGrade |  | 32 | Allow the program to store the physical education grade of a student | Pass |  |
| Program ran again |  |  |  |  |  |
| option | | | | | | integer | 4 | Allow the program to store and execute the user option from the menu to manage a class | Pass |  |
| option | integer | 2 | Allow the program to store and execute the user option from the menu to add a student under class management | Pass |  |
| tUsername | String | user | To allow program to store the name of the class teacher | Pass |  |
| tPassword | String | pass | To grant the user full access to the program | Pass |  |
| className | String | DemoClass | To allow program to store the name of the form teacher class | Pass |  |
| Student[i].fullName |  | Pan | Allow program store full name of a student | Pass |  |
| Student[i].mathGrade |  | 45 | Allow the program to store the math grade of a student | Pass |  |
| Student[i].engGrade |  | 67 | Allow the program to store the English grade of a student | Pass |  |
| Student[i].inteSciGrade |  | 87 | Allow the program to store the Integrated Science grade of the student | Pass |  |
| Student[i].phyEdGrade |  | 32 | Allow the program to store the physical education grade of a student | Pass |  |
| Program ran again |  |  |  |  |  |
| option | | | | | | integer | 3 | Allow the program to store and execute the user option from the menu to view a class report | Pass |  |
| tUsername | String | user | To allow program to store the name of the class teacher | Pass |  |
| tPassword | String | pass | To grant the user full access to the program | Pass |  |
| className | String |  | To allow program to store the name of the form teacher class | Pass |  |
| islogged |  | User/ pass | Ensure the login information is correct. | pass |  |
| readUsername | String | user | To ensure the right username is enter to allow the user access | Pass |  |
| readPassword | String | pass | To ensure the right password is enter to allow the user access | Pass |  |
| highestAverageStudent | String | Jan | To determine the name of the highest average student | Pass |  |
| heighestAverage | Integer | 52>0 | To determine the highest average of the students | Pass |  |
| heighestMathStudent | String | Jan | To determine the name of  The student with the highest Math average | Pass |  |
| heighestMath | Integer | 100>0 | To the determine the highest math average | Pass |  |
| highestEngStudent | String | Jan | To determine the name of  The student with the highest English average | Pass |  |
| heighestEng | integer | 29>0 | To the determine the highest English average | Pass |  |
| heighestInteSciStudent | String | Jan | To determine the name of  The student with the highest Integrated Science average | Pass |  |
| heighestInteSci | integer |  | To the determine the highest Integrated Science average | Pass |  |
| heighestPhysEdStudent | String | Jan | To determine the name of  the student with the highest Integrated Science average | Pass |  |
| heighestPhyEd | Integer | 39>0 | To the determine the highest physical education average | Pass |  |
| averageGrade | Real | 52 | To |  |  |
| ptr |  |  |  |  |  |
| fullName | String | Jan |  |  |  |
|  |  |  |  |  |  |

# **Screenshots**

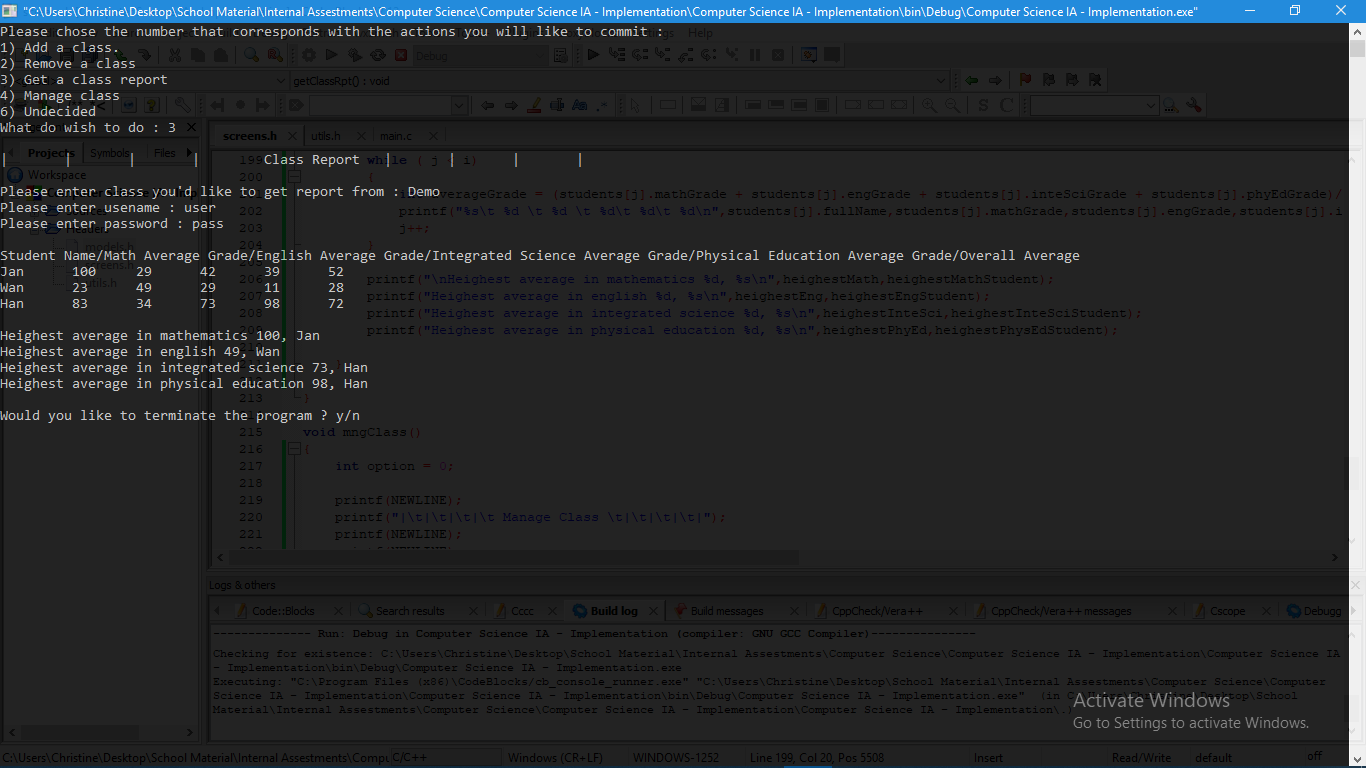


Figure 1.

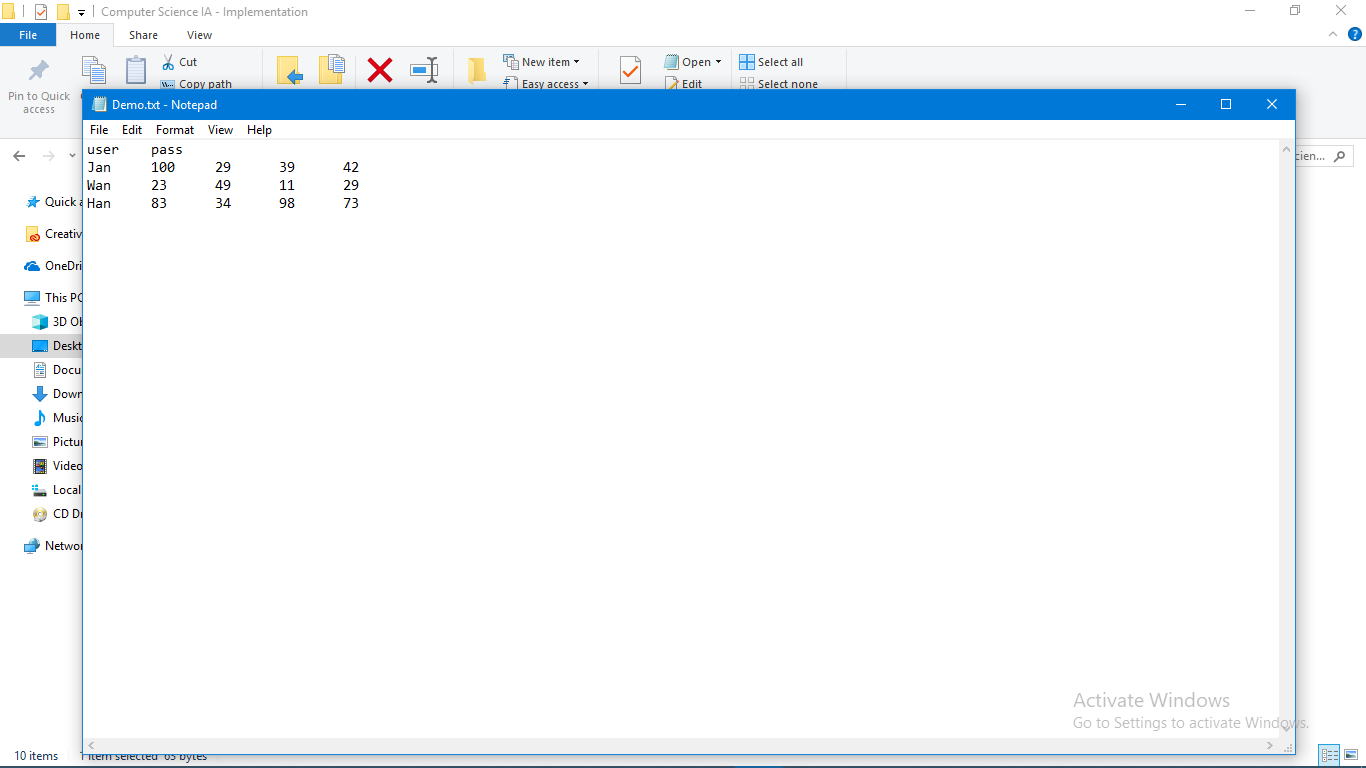


Figure 2

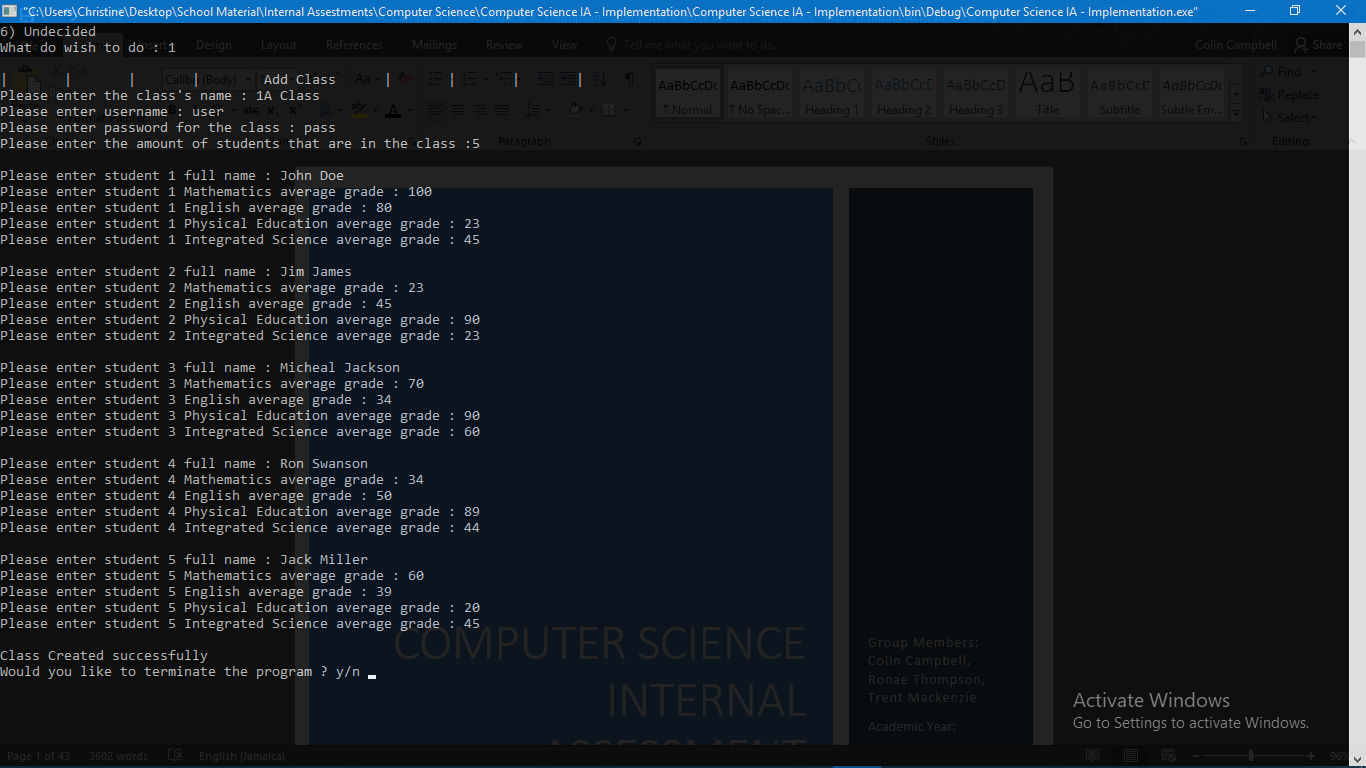


Figure 3