HANRÈ (WPJ) DELPORT

50605216

CMPG 121 CREATIVE PROJECT: PHASE 1

1. PROGRAM SCOPE

Wether it is to qualify you to apply for a job at your dream company, or to be considered for a big promotion, in this current era it's always necessary to keep on learning and acquiring new skills and knowledge.

Many young adults (and even older ones) attend universities and other similar establishments in hopes of being fully equipped with the needed knowledge and skills to enter the workforce, or to build their own start-ups. Although this may be one of the best ways (if you have the financial recourses to support you) to prepare you for your chosen career-path, you will still need various other skills, not taught by many of these big institutions.

This is why, for my Creative assignment, I decided to develop an application for a hypothetical educational establishment called "SkillRise". "SkillRise" is a tertiary educational establishment where students (of any age, profession or skill-level) can enrol in different short courses tailored toward teaching them new, high-in-demand skills and knowledge that will set them apart from their peers and colleagues.

The application's primary purpose is to enable students to view study-related information (such as their marks for each assessment, the different weights that each assessment counts toward final mark and a in-depth statistical analysis of the whole group's performance in the course) while enabling lecturers to manage students and courses (by creating, changing or adding students/courses).

The program will function as follows:

- The students will use the program to view all their marks for each subject and their overall mark for the whole course (marks can be viewed as soon as it is released). Each subject's final mark will consist of different types of assessments with different weights to all add up to the 100% (for e.g. The best 4 out of 6 quizzes may count 10%, while the best 3 out of 4 tests count 40% and the final exam counts 50% of the full mark). The calculation of each subject's final mark will be neatly displayed, showing all the marks the student obtained and the weight of each. Finally, a course average will be calculated and displayed. Students can also view statistics of the whole group (such as the average final mark, the standard deviation of the group, different quantiles of the group, etc) enabling them to measure their academic performance against others taking the same course, or similar courses
- The lecturers will be able to register new students and fill in the marks of existing students. They too will have access to the statistical analysis of the students' academic performance. Furthermore, they will be able to create new courses with the subjects available to "SkillRise" to teach.

I will make use of various techniques and tools that we have learned to use throughout this semester to ensure that the program meets all the requirements:

- I will use one array to store up to 70 students' information, and another one to store information related to courses.

- I will use 2 structs/classes. One struct will store all the needed information of the users, such as their name, surname, course signed up for (another struct) and unique, random generated student-ID. The other struct will save the course information, such as the course's name, a description of the course, the marks and the weights that each assessment counts toward final mark.
- I will make use of self-defined functions to simplify the programming process and to divide my program up into different smaller sections. I will write a function to display each menu when needed (it will receive an integer to indicate which menu to display). I will use another function to display all the statistics of the course, it will receive the course name and an array of users then run through the users to check who attends the course and to calculate the mean, min, max mark etc for the course. A third function will be used to register/create a new user and to save the data to the class and to a text file.
- One of my main focusses is to ensure that my program is user-friendly. The program will always have clear and concise output. User input will be standardised and clear explanations will be provided regarding the expected format for specific user input.
- Conditional statements and loops will be frequently used in my program for various tasks such as running the menu as long as user does not exit or checking if a student is registered by running through the array of registered students.
- I will use two text files. One for permanently storing the data of all registered users and another for storing the data of all courses. The "Users.txt" file will store the user's name, surname, generated ID ,course signed up for (if applicable), status of course (if the user finished the course or not), marks for the course etc. separated by "#"-symbol so it can be extracted and used to create objects in the main program. Similarly, the "Courses.txt" file will include the course's name, a description of the course and the assessment info of the specific course all seperated by the "#"-sign.
- I will use error traps (in the form of do-while loops) together with built-in functions such toupper(), tolower(), isdigit() etc to do input validation.
- I will replace the two structs mentioned above (the user struct and course struct) with two classes namely: "Users" and "Courses". The users class will describe objects with the same attributes as the variables mentioned in the struct description above(user's name, surname, generated ID ,course signed up for , status of ,marks for the course). The "Users" class will also define some methods for the initialization of the objects and methods to extract data from the object or to save data to the object. Examples of the methods may include a getName() to retrieve the name of a user from a declared object or a saveGrades() method to retrieve the grades from the main function and saving it to the object. Similarly, the "Courses" class will define objects with the same attributes as the "Courses" struct explained above. There will be different methods defined to create objects of the "Courses" class and to extract data from or save data to these objects. Methods to be used may include a getMarks() method to retrieve the full marks of the assessments of a specific course.
- I will use smart pointers when creating objects of the two classes mentioned above to avoid memory leaks. I will also make use of the different types of smart pointers where applicable, for

example by using a unique pointer when creating a student object or by using a weak pointer when accessing the data of a course.

- To create and use the smart pointers mentioned above, I will have to use generic templates defined in the “memory” header and in the std namespace, such as the unique_ptr<T>, shared_ptr<T> and weak_ptr<T> templates.

2. PSEUDOCODE

Main():
Display “Welcome to SkillRIse”

While user doesn’t exit program

Display “1. Sign in as Student
2. Sign in as Lecturer
3. Exit”
Receive input

Case choice:
1:
Ask user for unique student ID
If registered:
Display Welcome message including user’s information
While user doesn’t return:
Display : “1.View course details
2. View marks
3. View statistics
4. Return to main
Choose an option (1-4)”
Receive input
switch choice:
case 1: Display course details (such as duration and subjects and whether student have finished it or not)
case 2: Display student’s marks formatted neatly
case 3: Display statistics of specific course (such as the whole group’s mean, the standard deviation, the percentile in which the user falls and so forth)
case 4: Inform user they are returning to main menu
Default: Ask user to choose number between 1 and 4

Else if not registered
Display “You are not registered yet, enter 1 to register”
Receive input
if 1:
Receive Full Name
Receive Surname
Receive course that student will be studying

Generate student ID
Display Student ID and welcome student

Else:
Inform user that they will return to main menu
End case statement

2:
Ask lecturer for unique Username
If registered:
Display “Welcome ”user name , surname
While user doesn’t return:
Display: “1. Add course
2. Add Student
3. Edit student marks
4. View statistics
5. Return to main
Choose an option (1-5)”

Receive input
switch choice:
case 1: Receive and save info of new course
case 2: Receive and save info of new student
case 3: Receive mark to edit and save new mark (If all marks are entered, the student is finished with the course)
case 4: Receive specific course, and display it’s statistics
case 5: Inform lecturer they are returning to main menu
Default: Ask user to please enter number between 1 and 5

Else if not registered
Display “You are unfortunately not registered as a lecturer”
(user will be automatically returned to main menu)
End case statement

3:
Inform user they are about to exit;
End of case

Default:
Ask user to please enter number between 1 and 3

ShowStats(Student arrStudents[], string CourseName, float StudentAvg):

```
Counter = 0
For i==1 to arrStudentSize:
        TotalStudent = 0
        If arrStudents[i].CourseName = CourseName
                Counter++
                For j==1 to arrStudents[i].marks size:
                        Total += arrStudents[i].marks[j]

                AvgStudent = total/ (arrStudents[i].marks size)
                arrAverages[counter] = avgStudent
                TotalAll+=avgStudent
                TotalAllSquared += avgStudent^2

AvgAll = TotalAll/ arrStudents size
SDeviation = sqrt((1/( arrStudents size -1)) (TotalAllSquared-(arrStudents size(avgAll^2)))
Sort(arrAverages)

Q1 = arrAverages [round((arrStudentSize+1)*1/4)]
Q2 = arrAverages [round((arrStudentSize+1)*2/4)]
Q3 = arrAverages [round((arrStudentSize+1)*3/4)]
Min = arrAverages[0]
Max = arrAverages[counter]

Display: “The average for the ” CourseName “ course is: ” AvgAll “ with a standard Deviation of: ”
SDeviation
“The 5-point summary is as follows: ”
“Min: ” Min
“Q1: ” Q1
“Q2: ” Q2
“Q3: ” Q3
“Max: ” Max
“You have an average of ” StudentAvg
```

RegisterStudent(Student arrStudents[]):
Receive student's name and add to arrStudents
Receive student's surname and add to arrStudents
Receive the course that student wants to sign up for and add to arrStudents

Do
    Generate random 6 digit number
    For I == 1 to NumStudents
        If arrStudents[i].ID == generated num
        Num is not unique

    If num is unique and num is even (ID must be uneven to later identify user as a student)
    Num += 1
While generated number is not unique

Add generated ID to arrStudents

Write the new students info neatly to the "Users.txt" file in format:
Name#Surname#Course Name#Student ID

AddCourse(Course arrCourses[]):
Receive course name and add to arrCourses
Receive course description and add to arrCourses
Receive number of assessments

For I == 1 to num of assessments
    Receive mark of assessment
    Receive weight of assessment to the final mark
    Add each to their respective arrays in arrCourses

Write the new course info neatly to the "Courses.txt" file in format:
Course Name#Course Description#fullmark weight#fullmark weight#....

EditMarks(Student arrStudents[]):

```
For I == 1 to num of students
        Display arrStudents[i].ID
Receive the student ID

For I == 1 to num of students
        If arrStudents[i].ID == student ID
                Save pos


For I == 1 to num of marks
        Display arrStudents[pos].marks[i]

Receive the pos of mark that needs to be changed
Receive new mark
Save new mark in arrStudents[posStudent].marks[posMark]

Save students info in txtFile in format:
Name#Surname#Course Name#Student ID#mark1#mark2#mark3
```

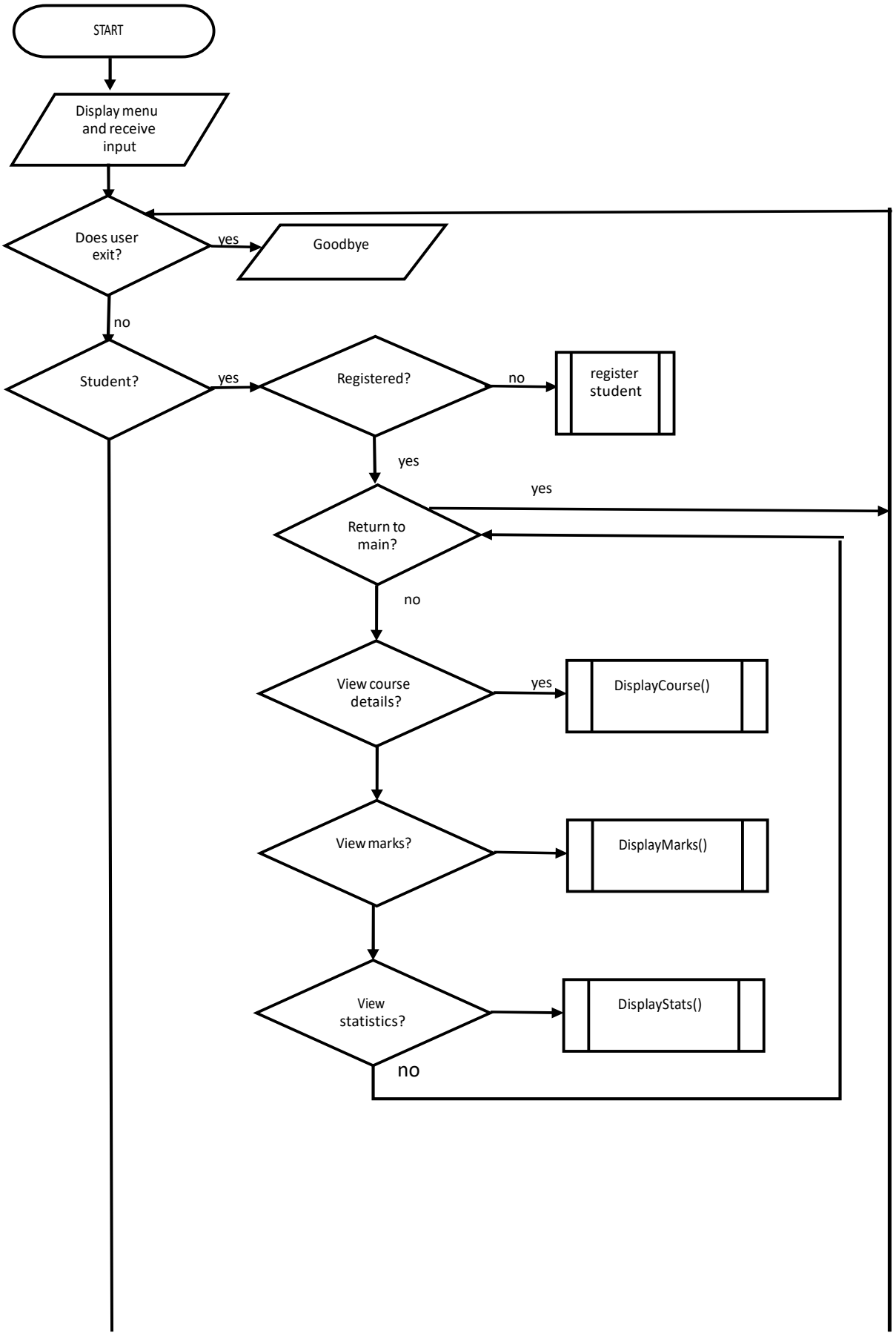
DisplayMarks(Student arrStudents[], Course arrCourses[] ,int studentID):

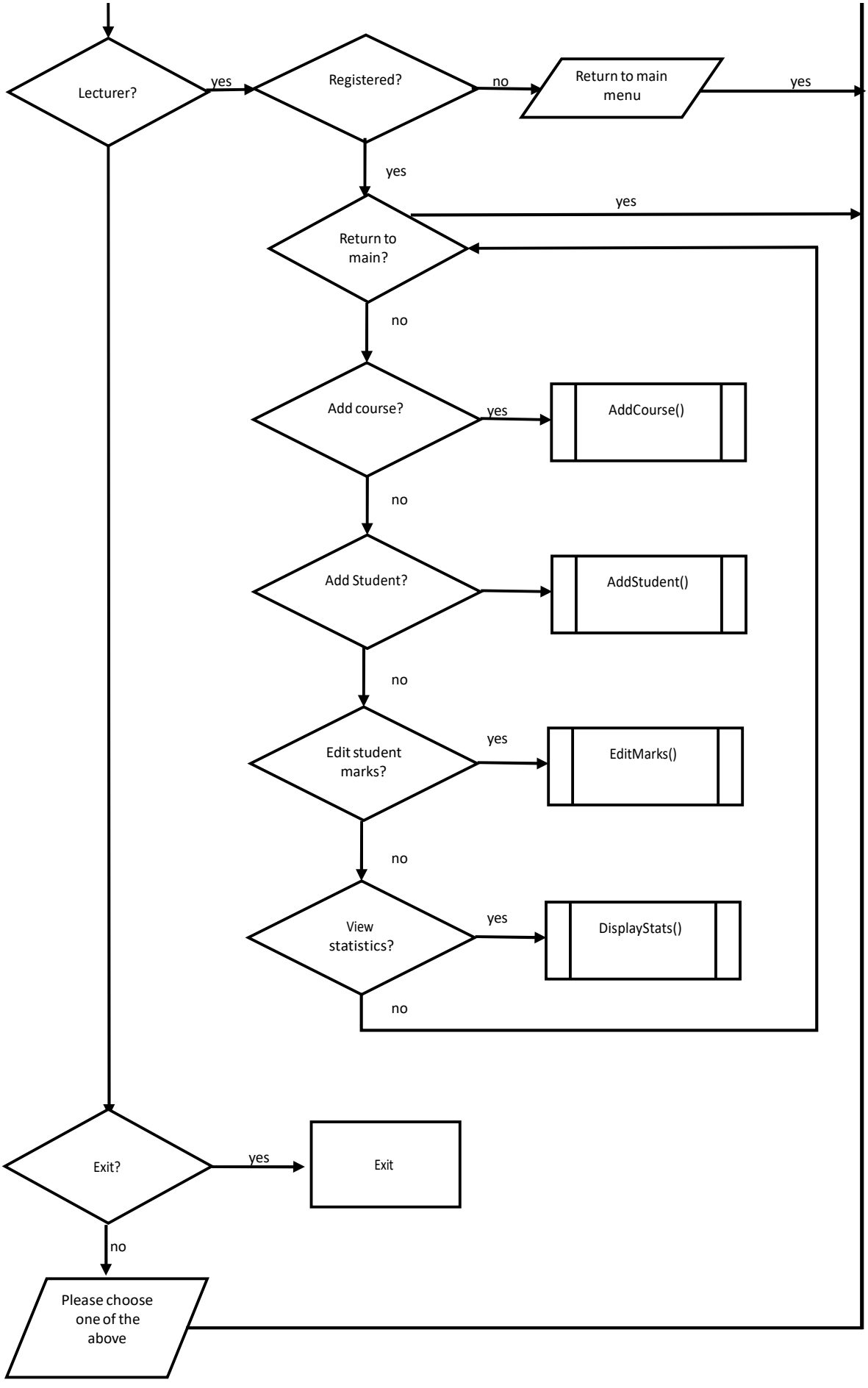
```
For I == 1 to num of students
        If arrStudents[i].ID == student ID
                Save pos

For i==1 to num of courses
        If arrCourses[i].name == arrStudents[pos].courseName
        Save coursePos

Num of marks =  Sizeof arrCourses[coursePos].marks[]

For I == 1 to num of marks

        Display arrStudents[pos].marks[i]
```

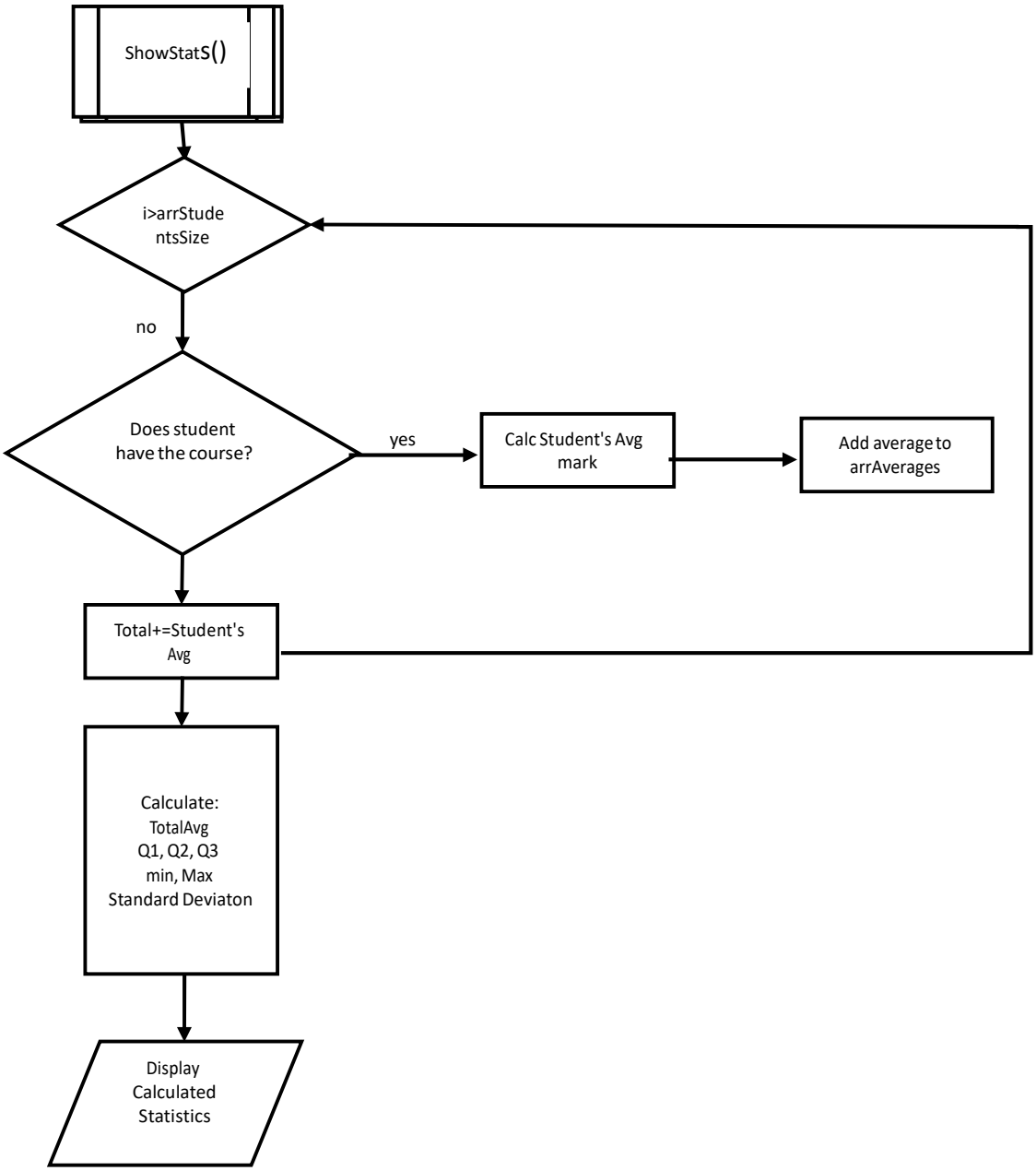
DisplayCourse(Course arrCourses[] ,string CourseName):

```
For i==1 to num of courses
        If arrCourses[i].name == CourseName
        Save coursePos
Display arrCourses[coursePos].name
Display arrCourses[coursePos].description

For I == 1 to num of marks
        Display arrCourses[i].marks[i]          arrCourses[i].weights[i]
```

3.FLOWCHARTS




Lecturer?
yes
Registered?
no
Return to main menu
yes
yes
yes
Return to main?
no
Add course?
yes
AddCourse()
no
Add Student?
AddStudent()
no
Edit student marks?
yes
EditMarks()
no
View statistics?
yes
DisplayStats()
no
Exit?
yes
Exit
no
Please choose one of the above


ShowStatS()
i>arrStude
ntsSize
no
Does student
have the course?
yes
Calc Student's Avg
mark
Add average to
arrAverages
Total+=Student's
Avg
Calculate:
TotalAvg
Q1, Q2, Q3
min, Max
Standard Deviaton
Display
Calculated
Statistics

4. RESEARCH ON ADVANCED REQUIREMENTS

Classes:

According to NIU (n.d : para 1) a class describes a programming object by giving it a list of attributes (variable declarations) and defining member functions that a object of the class will use. When you define a class , you actually define a new data type, just like with a struct.

I plan to use two classes. One class for the users and another class for the different courses.

The attributes of the "user" class will include: Name, surname, Generated ID (that will also indicate if he/she is a student or a lecturer), course signed up for, status of course (if the user finished the course or not) and the marks for the course. The methods will include a constructer method (to initialize and create a new member of the class) and different mutator and accessor methods such as methods to retrieve the name and surname of the user (getNames), a method to save the marks of the student (saveMarks), a method to save and generate a unique ID for the student (generateID) etc.

Each course will have different attributes, such as the Course name, a description of the course and the assessment info of the specific course. Each course will have 2 exams that need to count 60% of the final mark, the rest of the assessments will count 40% of the final mark. The full marks of the assessments will be stored in an array, with the marks of the two exams at the end (such as [30, 30, 30, 30, 60, 60]). The "Course" class will have a constructor method (to initialize and create a new course), a method to retrieve the name and description of the course, and a method that will return an array of the marks that the assessments count, to the main program (that will be used to calculate the average of the student by dividing the student's mark for each assessment by the marks that the assessment count of and multiply it by the final weights of the assessments).

Smart Pointers and Generic Templates

Smart pointers, like raw pointers, store an address but they are mostly used for dynamically allocating memory (Horton, 2014:173). Smart pointers are useful tools to avoid "memory leaks" because they deallocate memory automatically.

The generic templates, defining the different types of smart pointers are defined in the *memory* header and in the std namespace . (Horton, 2014:173). According to Horton (2014:17) the three types of smart pointers that are defined in the std namespace is:

- Unique pointers (with the template: unique_ptr<T>)
- Shared pointers (with the template: shared_ptr<T>)
- Weak pointers (with the template: weak_ptr<T>)

In my application I will make use of smart pointers when creating objects of the two classes mentioned above to avoid memory leaks. I will also make use of the different types of smart pointers where applicable, for example by using a unique pointer when creating a student object or by using a weak

pointer when accessing the data of a course. When creating, accessing or editing objects of the classes, I will use smart pointers instead of variables.

REFERENCE LIST

NIU (Northen Illinois University) Department of Computer Science. n.d . *Classes And Objects* https://faculty.cs.niu.edu/~mcmahon/CS241/Notes/classes_and_objects.html Date of access: 25 Sept. 2024.

Horton, I. 2014. *Beginning C++.* 1st ed. New-York, United States. Apress.