****

**COMSATS University Islamabad, Lahore Campus**

**Programming Fundamentals (CSC 103)**

**LAB PROJECT**

**ANNEX\_1**

**PROGRAM SPECIFICATION**

**DEVELOPED BY:**

Muhammad Ahmad (FA23-BCE-113)

**PROGRAM NAME**

“*Student Management System with Fine Calculation”*

* **Purpose or problem definition:**

The purpose of this project is to discuss about Student Registration with fine. The system aims to manage student registrations, course enrollments, and handle late registrations with fine calculations. The project's objective is to incorporate key programming concepts such as if-else statements, loops, arrays, structures, and functions to create an efficient and user-friendly system.

* **Program Procedures**

**1. Student Registration:**

1. **Input:** Student details (ID, name, contact information).
2. **Process:** Add new student to the system, update student count.
3. **Output:** Confirmation message.

**2. Course Enrollment:**

1. **Input:** Student ID, course details, enrollment date.

2. **Process:** Enroll student in a course, update enrollment count.

3. **Output:** Confirmation message.

**3. Late Enrollment Fine:**

1. **Input:** Late enrollment in course details (days late).

2. **Process:** Calculate fine based on predefined rate, update fine information.

3. **Output:** Display and update fines for each late-registered student.

**4. User Interface:**

1. **Input:** User choices through a menu system.

2.  **Process:** Utilize if-else statements to handle different user choices and cases.

3. **Output:** Display relevant information and interface updates.

**5. Data Persistence:**

1. **Input:** Option to save/load data from a file.

2. **Process:** Implement functionality for saving and loading student, enrollment, and fine data.

3. **Output:** Confirmation message for successful data persistence.

* **Pseudo Code:**
* procedure **main**()

// Display the program's decorative header

**displayDesign1()**

// Load student data from the file

**loadFromFile()**

// Declare a variable for user input

**int** choice

// Main menu **loop**

repeat

**displayMainMenu**()

// Get user input for menu choice

input choice

// Switch statement to execute the selected menu option

**switch** choice

**case** **1**:

// Add a new student

input id, name, contact

addStudent(id, name, contact)

**case 2:**

// Display registered students

displayStudents()

**case** **3**:

// Enroll a student in a course

input studentID, courseID, enrollmentDate

displayCourseList()

enrollStudent(studentID, courseID, enrollmentDate)

**case** **4**:

// Display courses enrolled by a student

input studentID

displayStudentCourses(studentID)

**case** **5**:

// Calculate fines for a student

input studentID

**float** fine = calculateFine(studentID)

output "Fine for Student ID " + studentID + ": $" + fine

**case** **6**:

// Display fines for all students

**displayFines()**

**case** **7**:

// Save student data to file

**saveToFile()**

**case** **8**:

// Load student data from file

**loadFromFile()**

**case** **9**:

output "Exiting..."

otherwise:

output "Invalid choice! Please enter a number between 1 and 9."

until choice **is** **9** // Continue the loop until the user chooses to exit

end procedure

* procedure **displayMainMenu**()

output "=====================================|"

output "| STUDENT MANAGEMENT |"

output "|====================================|"

output "| 1. Add Student |"

output "| 2. Display Registered Students |"

output "| 3. Enroll Student in a Course |"

output "| 4. Display Student's Courses |"

output "| 5. Calculate Fines |"

output "| 6. Display Fines |"

output "| 7. Save Data to File |"

output "| 8. Load Data from File |"

output "| 9. Exit |"

output "|====================================="

output ""

output "Enter your choice: "

* procedure **displayCourseList**()

output "Available Courses:"

**for** each course **in** courses

output course.courseID, course.courseName

end **for**

* procedure **addStudent**(id, name, contact)

// Create a new student object

student = **new** Student()

student.studentID = id

student.studentName = name

student.contactInfo = contact

// Add the student to the list of students

students.**add**(student)

output "Student added successfully."

* procedure **displayStudents**()

output "Registered Students:"

**for** each student **in** students

output student.studentID, student.studentName, student.contactInfo

end **for**

* procedure **enrollStudent**(studentID, courseID, enrollmentDate)

// Find the student with the given ID

student = findStudentByID(studentID)

// Find the course with the given ID

course = findCourseByID(courseID)

// Create an enrollment record

enrollment = **new** Enrollment()

enrollment.courseID = courseID

enrollment.enrollmentDate = enrollmentDate

// Add the enrollment to the student's record

student.enrollments.**add**(enrollment)

output "Student enrolled in the course successfully."

* procedure **displayStudentCourses**(studentID)

// Find the student with the given ID

student = findStudentByID(studentID)

output "Courses enrolled by Student ID", studentID, ":"

**for** each enrollment **in** student.enrollments

course = findCourseByID(enrollment.courseID)

output course.courseID, course.courseName, enrollment.enrollmentDate

end **for**

* function **calculateFine**(studentID)

// Find the student with the given ID

student = findStudentByID(studentID)

// Calculate fine based on your specific logic

fine = **0**

**for** each enrollment **in** student.enrollments

// Calculate fine for the course and add to total fine

fine = fine + calculateFineForCourse(enrollment.courseID)

end **for**

**return** fine

* procedure **displayFines**()

output "Fines for all students:"

**for** each student **in** students

fine = calculateFine(student.studentID)

output "Student ID", student.studentID, "Fine: $", fine

end **for**

* procedure **saveToFile**()

// Save student data to a file using your preferred file format

* procedure **loadFromFile**()

// Load student data from a file using your preferred file format

* **Step-by-Step Instructions:**

1. **Open the Program:**
   * Open the student management system application.
2. **Display Menu:**
   * Observe the main menu presented on the screen.
   * Choose an option by entering the corresponding number.
3. **Add a New Student:**
   * Select the option to add a new student.
   * Provide the required details such as Student ID, name, and contact information.
   * Confirm to add the student to the system.
4. **Display Registered Students:**
   * Select the option to display students option.
   * View a table showing registered students' information (ID, name, contact).
5. **Enroll Student in a Course:**
   * Choose the option to enroll a student in a course.
   * Input the Student ID, course details, and enrollment date.
   * Verify and enroll the student in the specified course.
6. **Display Student Courses:**
   * Select the option to display courses enrolled by a student.
   * Enter the Student ID to view the list of courses taken by that student.
7. **Calculate Fines:**
   * If applicable, select the option to calculate fines.
   * Follow on-screen instructions to calculate fines for late returns.
8. **Save Student Data:**
   * Choose the option to save student data to a file.
   * Confirm the save operation and receive a success or failure message.
9. **Load Student Data:**
   * Opt for loading student data from a file.
   * Confirm the load operation and check for success or failure feedback.
10. **Exit the System:**
    * Choose the exit option to close the student management system.

* **NOTES & RESTRICTIONS:**
* The system currently has a limitation of managing up to 100 students, fixed maximum number of courses per student (5).We can change it by just changing the constant.

.

* The program assumes that the user provides correct inputs and does not incorporate extensive input validation.
* Date should be in proper format in course enrollment.
* **DRAWBACKS:**
* The program add more than one student with same ID.
* Simple array (pre-defined size) is declare instead of dynamic (manage its size at run time)
* Student enroll in same course more than one time at the same moment.
* Data that want to be safe on a file is in order of bottom to top instead of top to bottom.
* There is no proper function for clearing data from file.
* Display on console page may not follow symmetry i.e. the use of proper set width (setw)

was neglected. .

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ANNEX\_2**

USE CASE DEFINATION

**DEVELOPED BY:**

Muhammad Ahmad (FA23-BCE-113)

**PUBLISHED DATE:** 1 January, 2024

**PROGRAM NAME**

“*Student Management System with Fine Calculation”*

* **Use Case 1: Add a New Student**

**Goal:** To add a new student to the system.  
**Steps:**

1. User selects the "Add Student" option from the main menu.
2. User enters the student's ID, name, and contact information.
3. The system validates the input and adds the student to the database.
4. System displays a success message if the addition is successful.
5. User returns to the main menu.

* **Use Case 2: Display Registered Students**

**Goal:** To view a list of all registered students.  
**Steps:**

1. User selects the "Display Registered Students" option from the main menu.
2. The system displays the list of registered students.
3. User views the student IDs, names, and contact information.
4. User returns to the main menu.

* **Use Case 3: Enroll Student in a Course**

**Goal:** To enroll a student in a course.  
**Steps:**

1. User selects the "Enroll Student in a Course" option from the main menu.
2. User enters the student's ID.
3. The system displays the list of available courses.
4. User selects a course to enroll the student.
5. User enters the enrollment date.
6. The system validates the input and enrolls the student in the course.
7. System displays a success message if the enrollment is successful.
8. User returns to the main menu.

* **Use Case 4: Display Student's Courses**

**Goal:** To view the courses in which a student is enrolled.  
**Steps:**

1. User selects the "Display Student's Courses" option from the main menu.
2. User enters the student's ID.
3. The system retrieves and displays the list of courses in which the student is enrolled.
4. User views course IDs, names, and enrollment dates.
5. User returns to the main menu.

* **Use Case 5: Calculate Fines**

**Goal:** To calculate fines for a student based on late enrollment.  
**Steps:**

1. User selects the "Calculate Fines" option from the main menu.
2. User enters the student's ID.
3. The system calculates fines for late enrollment in each course.
4. System displays the total fine for the student.
5. User returns to the main menu.

* **Use Case 6: Display Fines**

**Goal:** To view fines for all students with late enrollments.  
**Steps:**

1. User selects the "Display Fines" option from the main menu.
2. The system calculates fines for all students.
3. System displays a list of students with fines, including their IDs, names, and fines.
4. User returns to the main menu.

* **Use Case 7: Save Data to File**

**Goal:** To save student and enrollment data to a file.  
**Steps:**

1. User selects the "Save Data to File" option from the main menu.
2. The system writes student and enrollment data to a file named "info.txt."
3. System displays a success message.
4. User returns to the main menu.

* **Use Case 8: Load Data from File**

**Goal:** To load previously saved student and enrollment data from a file.  
**Steps:**

1. User selects the "Load Data from File" option from the main menu.
2. The system reads student and enrollment data from the "info.txt" file.
3. System displays a success message.
4. User returns to the main menu.

* **Use Case 9: Exit**

**Goal:** To exit the program.  
**Steps:**

1. User selects the "Exit" option from the main menu.
2. The system terminates, and the program closes.

These use cases cover various interactions users might have with the student management system. Each use case describes the user's goal, the steps involved, and the expected outcomes.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ANNEX\_3**

**PROGRAM CODE**

**DEVELOPED BY:**

Muhammad Ahmad (FA23-BCE-113)

**PROGRAM NAME**

“*Student Management System with Fine Calculation”*

#include <fstream> // for file handling

#include <string> // for including string

#include <iomanip> // for formating

#include <cmath> // for mathematical function

#include <iostream> // for cin and cout

**using** **namespace** std;

// Constants

**const** **int** MAX\_COURSES = **5**;

**const** **float** FINE\_RATE = **5**;

**const** string LAST\_DATE\_TO\_ENROLL = "01/01/2024"; // change with requirment

// Structure to represent a course

**struct** Course

{

**int** courseID;

string courseName;

};

// Number of courses and course information

**const** **int** NUM\_COURSES = **5**; // change with requirment

Course courses[NUM\_COURSES] =

{

{**1**, "Applied Physics"},

{**2**, "Calculus"},

{**3**, "Programming Fundamentals"},

{**4**, "Functional English"},

{**5**, "Ideology and Constitution"},

};

// Structure to represent a course enrollment

**struct** CourseEnrollment

{

**int** courseID;

string enrollmentDate;

};

// Structure to represent a student

**struct** Student

{

**int** studentID;

string name;

string contact;

**int** totalCourses;

CourseEnrollment courses[MAX\_COURSES];

};

// Array to store student information

Student students[**100**];

**int** totalStudents = **0**;

// Function prototypes

**void** **displayCourseList**();

**void** **enrollStudent**(**int** studentID, **int** courseID, string enrollmentDate);

**int** **daysInMonth**(**int** month);

**void** **displayStudentCourses**(**int** studentID);

**int** **differenceInDays**(**const** string &enrollmentDate);

**float** **calculateFine**(**int** studentID);

**void** **displayFines**();

**void** **addStudent**(**int** id, string name, string contact);

**void** **displayStudents**();

**bool** **fileExists**(**const** string &filename);

**void** **createFile**(**const** string &filename);

**void** **loadFromFile**();

**void** **saveToFile**();

**void** **displayDesign1**();

**int** **main**()

{

// Display the program's decorative header

displayDesign1();

// Load student data from the file

loadFromFile();

// Declare a variable for user input

**int** choice;

// Main menu loop

**do**

{

cout << "**\n**";

cout << "=====================================|" << endl;

cout << "| STUDENT MANAGEMENT |" << endl;

cout << "|====================================|" << endl;

cout << "| 1. Add Student |" << endl;

cout << "| 2. Display Registered Students |" << endl;

cout << "| 3. Enroll Student in a Course |" << endl;

cout << "| 4. Display Student's Courses |" << endl;

cout << "| 5. Calculate Fines |" << endl;

cout << "| 6. Display Fines |" << endl;

cout << "| 7. Save Data to File |" << endl;

cout << "| 8. Load Data from File |" << endl;

cout << "| 9. Exit |" << endl;

cout << "|=====================================" << endl;

cout << "**\n**";

cout << "Enter your choice: ";

cin >> choice;

cout << "**\n**";

// Switch statement to execute the selected menu option

**switch** (choice)

{

**case** **1**:

{

// Add a new student

**int** id;

string name, contact;

cout << "Enter student ID: ";

cin >> id;

cout << "Enter student name: ";

cin.ignore();

getline(cin, name);

cout << "Enter contact information: ";

getline(cin, contact);

addStudent(id, name, contact);

**break**;

}

**case** **2**:

// Display registered students

displayStudents();

**break**;

**case** **3**:

{

// Enroll a student in a course

**int** studentID, courseID;

string enrollmentDate;

cout << "Enter student ID: ";

cin >> studentID;

// Display the list of available courses

displayCourseList();

cout << "Enter course ID to enroll: ";

cin >> courseID;

cout << "Enter enrollment date (DD/MM/YYYY): ";

cin.ignore();

getline(cin, enrollmentDate);

enrollStudent(studentID, courseID, enrollmentDate);

// Save enrollment information in the student's struct

students[totalStudents - **1**].courses[students[totalStudents - **1**].totalCourses - **1**].courseID = courseID;

students[totalStudents - **1**].courses[students[totalStudents - **1**].totalCourses - **1**].enrollmentDate = enrollmentDate;

**break**;

}

**case** **4**:

{

// Display courses enrolled by a student

cout << "**\n**";

**int** studentID;

cout << "Enter student ID: ";

cin >> studentID;

displayStudentCourses(studentID);

**break**;

}

**case** **5**:

{

// Calculate fines for a student

**int** studentID;

cout << "Enter student ID: ";

cin >> studentID;

**float** fine = calculateFine(studentID);

cout << "Fine for Student ID " << studentID << ": $" << fine << endl;

**break**;

}

**case** **6**:

// Display fines for all students

displayFines();

**break**;

**case** **7**:

// Save student data to file

saveToFile();

**break**;

**case** **8**:

// Load student data from file

loadFromFile();

**break**;

**case** **9**:

cout << "Exiting..." << endl;

**break**;

**default:**

cout << "Invalid choice! Please enter a number between 1 and 9." << endl;

**break**;

}

}

**while** (choice != **9**); // Continue the loop until the user chooses to exit

**return** **0**;

}

// --------------------------------------------------------------------------

// Function to display the list of available courses

**void** **displayCourseList**()

{

cout << "Available Courses:" << endl;

**for** (**int** i = **0**; i < NUM\_COURSES; ++i)

{

cout << "ID: " << courses[i].courseID << ", Name: " << courses[i].courseName << endl;

}

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// Function to enroll a student in a course

**void** **enrollStudent**(**int** studentID, **int** courseID, string enrollmentDate)

{

// Iterate through students to find the matching student ID

**for** (**int** i = **0**; i < totalStudents; ++i)

{

// Check if student ID matches

**if** (students[i].studentID == studentID)

{

// Check if the student has not reached the maximum number of courses

**if** (students[i].totalCourses < MAX\_COURSES)

{

// Add course information to the student's record

students[i].courses[students[i].totalCourses].courseID = courseID;

students[i].courses[students[i].totalCourses].enrollmentDate = enrollmentDate;

students[i].totalCourses++;

cout << "Student enrolled in the course successfully!" << endl;

**return** ;

}

**else**

{

cout << "Student has already enrolled in the maximum number of courses." << endl;

**return** ;

}

}

}

cout << "Student not found." << endl;

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// Function to display the courses enrolled by a student

**void** **displayStudentCourses**(**int** studentID)

{

// Iterate through students to find the matching student ID

**for** (**int** i = **0**; i < totalStudents; ++i)

{

// Check if student ID matches

**if** (students[i].studentID == studentID)

{

cout << "Courses enrolled by Student ID " << studentID << ":" << endl;

// Iterate through the courses and display information

**for** (**int** j = **0**; j < students[i].totalCourses; ++j)

{

**int** courseID = students[i].courses[j].courseID;

string courseName;

// Find the course name based on the courseID

**for** (**int** k = **0**; k < NUM\_COURSES; ++k)

{

**if** (courses[k].courseID == courseID)

{

courseName = courses[k].courseName;

**break**;

}

}

cout << "Course ID: " << courseID << ", Course Name: " << courseName << ", Enrollment Date: " << students[i].courses[j].enrollmentDate << endl;

}

**return**;

}

}

cout << "Student not found." << endl;

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

//Function to return days of specific month

**int** **daysInMonth**(**int** month)

{

// 1st index is 0 so that the month match with index

**int** daysInMonth[] = {**0**, **31**, **28**, **31**, **30**, **31**, **30**, **31**, **31**, **30**, **31**, **30**, **31**};

**return** daysInMonth[month];

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// Function to calculate the difference in days between two dates

**int** **differenceInDays**(**const** string &enrollmentDate)

{

**int** day1, month1, year1;

**int** day2, month2, year2;

**int** diff;

// Extract day, month, and year from the enrollment date and the last date to enroll

sscanf(enrollmentDate.c\_str(), "%d/%d/%d", &day1, &month1, &year1);

sscanf(LAST\_DATE\_TO\_ENROLL.c\_str(), "%d/%d/%d", &day2, &month2, &year2);

// Calculate total days for both dates

**int** days1 = year1 \* **365** + day1;

**for** (**int** i = **1**; i <= month1; ++i)

{

days1 += daysInMonth(i);

}

**int** days2 = year2 \* **365** + day2;

**for** (**int** i = **1**; i <= month2; ++i)

{

days2 += daysInMonth(i);

}

// Calculate the difference in days

diff = (days1 - days2);

**return** diff;

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// Function to calculate the fine for a student

**float** **calculateFine**(**int** studentID)

{

**float** totalFine = **0.0**;

// Iterate through students to find the matching student ID

**for** (**int** i = **0**; i < totalStudents; ++i)

{

// Check if student ID matches

**if** (students[i].studentID == studentID)

{

// Iterate through the courses enrolled by the student

**for** (**int** j = **0**; j < students[i].totalCourses; ++j)

{

**int** daysDifference = differenceInDays(students[i].courses[j].enrollmentDate);

// Check if enrollment is late

**if** (daysDifference > **0**)

{

totalFine += FINE\_RATE \* daysDifference;

}

}

**break**;

}

}

**return** totalFine;

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// Function to display fines for all students

**void** **displayFines**()

{

cout << "Student Fines:" << endl;

**bool** foundFines = false;

// Iterate through students to calculate and display fines

**for** (**int** i = **0**; i < totalStudents; ++i)

{

**float** fine = calculateFine(students[i].studentID);

**if** (fine > **0.0**)

{

cout << "Student ID: " << students[i].studentID << ", Name: " << students[i].name << ", Fine: $" << fine << endl;

foundFines = true;

}

}

// Display a message if no fines are found

**if** (!foundFines)

{

cout << "No fines found for any student." << endl;

}

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// Function to add a new student to the system

**void** **addStudent**(**int** id, string name, string contact)

{

// Check if there is space for a new student

**if** (totalStudents < **100**)

{

// Assign values to the new student

students[totalStudents].studentID = id;

students[totalStudents].name = name;

students[totalStudents].contact = contact;

students[totalStudents].totalCourses = **0**; // Initialize total courses to 0 initially.

// Increment the total number of students

totalStudents++;

cout << "Student added successfully!" << endl;

}

**else**

{

cout << "Maximum number of students reached!" << endl;

}

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// Function to display the list of all registered students

**void** **displayStudents**()

{

cout << "Registered Students:" << endl;

**for** (**int** i = **0**; i < totalStudents; ++i)

{

cout << "ID: " << students[i].studentID << ", Name: " << students[i].name << ", Contact: " << students[i].contact << endl;

}

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// Function to check if a file exists

**bool** **fileExists**(**const** string &filename)

{

ifstream file(filename);

**return** file.good();

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// Function to create a new file

**void** **createFile**(**const** string &filename)

{

ofstream outfile(filename);

**if** (!outfile)

{

cout << "Error creating file '" << filename << "'." << endl;

**return**;

}

cout << "File '" << filename << "' created successfully." << endl;

outfile.close();

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

**void** **loadFromFile**()

{

// Check if the file 'info.txt' exists

**if** (!fileExists("info.txt"))

{

cout << "File 'info.txt' does not exist." << endl;

**return**;

}

// Open the file for reading

ifstream inputFile("info.txt");

**if** (!inputFile)

{

cout << "Unable to open file 'info.txt' for reading." << endl;

**return**;

}

// Reset the total number of students to 0

totalStudents = **0**;

// Read student data from the file

**while** (inputFile >> students[totalStudents].studentID >> students[totalStudents].name >> students[totalStudents].contact >> students[totalStudents].totalCourses)

{

// Read course enrollment information

**for** (**int** j = **0**; j < students[totalStudents].totalCourses; ++j)

{

inputFile >> students[totalStudents].courses[j].courseID >> students[totalStudents].courses[j].enrollmentDate;

}

// Increment the total number of students

totalStudents++;

}

// Close the input file

inputFile.close();

cout << "Data loaded from file successfully." << endl;

}

// --------------------------------------------------------------------------

// --------------------------------------------------------------------------

// Function to save student data to a file

**void** **saveToFile**()

{

// Open the file for writing

ofstream outputFile("info.txt");

**if** (!outputFile)

{

cout << "Unable to open file 'info.txt' for writing." << endl;

**return**;

}

// Write student data to the file

**for** (**int** i = **0**; i < totalStudents; ++i)

{

// Write student information

outputFile << students[i].studentID << " " << students[i].name << " " << students[i].contact << " " << students[i].totalCourses << "**\n**";

// Write course enrollment information

**for** (**int** j = **0**; j < students[i].totalCourses; ++j)

{

outputFile << students[i].courses[j].courseID << " " << students[i].courses[j].enrollmentDate << "**\n**";

}

}

// Close the output file

outputFile.close();

cout << "Data saved to file successfully." << endl;

}

// ------------------------------------------------------------------------

// ------------------------------------------------------------------------

// Function to display a decorative header

**void** **displayDesign1**()

{

cout << "**\n**";

cout << setw(**75**) << setfill(' ') << " \_-------------------------------------------\_ " << endl;

cout << setw(**75**) << setfill(' ') << " -:- Welcome to STUDENT MANAGEMENT -:- " << endl;

cout << setw(**75**) << setfill(' ') << " -:\*:- Registration System -:\*:- " << endl;

cout << setw(**75**) << setfill(' ') << " -:\*:\*:- -:\*:\*:- " << endl;

cout << setw(**75**) << setfill(' ') << " -:\*:\*:\*:- === Developed by === -:\*:\*:\*:- " << endl;

cout << setw(**75**) << setfill(' ') << " -:\*:\*:- -:\*:\*:- " << endl;

cout << setw(**75**) << setfill(' ') << " -:\*:- MUHAMMAD AHMAD & ABDULLAH ASHRAF -:\*:- " << endl;

cout << setw(**75**) << setfill(' ') << " -:- FA23-BCE-113 | FA23-BCE-118 -:- " << endl;

cout << setw(**75**) << setfill(' ') << " -\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_- " << endl;

cout << "**\n**" ;

}

// ------------------------------------------------------------------------