

**WORKSHEET 2**

**Submitted By: Sandhya Aryal**

**Student ID: 24000860**

**Cyber Security and Digital Forensics**GitHub Link:

<https://github.com/Sandhyaaaa1/Cpp_Worksheet>

**Task 1: Basic student grading system prototype using classes and objects.**

**[30 Marks]**

Write a program that manages a simple student grade calculator with the following requirements. Create a Student class that has:

1. Student name (string)
2. Three subject marks (integers)
3. A basic member function to calculate average

#include <iostream>

#include <string>

using namespace std;

class Student

{

private:

string name;

int marks[3];

public:

void input\_Details()

{

cout << "Enter the name of students: ";

cin >> name;

for (int i = 0; i < 3; i++)

{

cout << "Enter marks of subject " << i + 1 << ": ";

cin >> marks[i];

while (marks[i] < 0 || marks[i] > 100)

{

cout << "Your input is Invalid! The marks must be between 0 and 100. Please re-enter: ";

cin >> marks[i];

}

}

}

int calculate\_Total()

{

return marks[0] + marks[1] + marks[2];

}

double calculate\_Average()

{

return static\_cast<double>(calculate\_Total()) / 3;

}

char calculate\_Grade()

{

double average = calculate\_Average();

if (average >= 90)

{

return 'A';

}

else if (average >= 80)

{

return 'B';

}

else if (average >= 70)

{

return 'C';

}

else if (average >= 60)

{

return 'D';

}

else

{

return 'F';

}

}

void display\_Results()

{

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

cout << "Total Marks: " << calculate\_Total() << endl;

cout << "Average Marks: " << calculate\_Average() << "%" << endl;

cout << "Grade: " << calculate\_Grade() << endl;

}

};

void Student\_Grading()

{

Student s1;

s1.input\_Details();

s1.display\_Results();

}

int main()

{

Student\_Grading();

return 0;

}

A screenshot of a computer

AI-generated content may be incorrect.

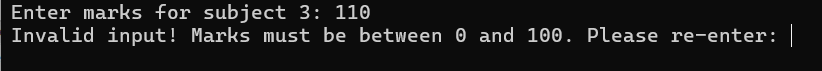
The program should:

1. Accept student details (name and marks) from user input
2. Calculate and display:
   1. Total marks
   2. Average marks
   3. Grade (A for ≥90%, B for ≥80%, C for ≥70%, D for ≥60%, F for <60%)

A black screen with white text

AI-generated content may be incorrect.

1. Display a message if any mark is below 0 or above 100

****

**Task 2: Programming assignments: All questions are mandatory**

1. **Write a program with a class Circle having:**
   1. **Private member: radius (float)**
   2. **A constructor to initialize radius**
   3. **A friend function compareTwoCircles that takes two Circle objects and prints which circle has the larger area**

#include <iostream>

#include <cmath>

using namespace std;

class Circle;

void compareTwoCircles(Circle &c1, Circle &c2);

class Circle

{

private:

float radius;

public:

Circle(float r)

{

radius = r;

}

float area()

{

return M\_PI \* radius \* radius;

}

friend void compareTwoCircles(Circle &c1, Circle &c2);

};

void compareTwoCircles(Circle &c1, Circle &c2)

{

float area1 = c1.area();

float area2 = c2.area();

cout << "Area of Circle 1: " << area1 << endl;

cout << "Area of Circle 2: " << area2 << endl;

if (area1 > area2)

{

cout << "Circle 1 has the larger area." << endl;

}

else if (area1 < area2)

{

cout << "Circle 2 has the larger area." << endl;

}

else

{

cout << "Both circles have the same area." << endl;

}

}

int main()

{

float radius1, radius2;

cout << "Enter radius of Circle 1: ";

cin >> radius1;

cout << "Enter radius of Circle 2: ";

cin >> radius2;

Circle circle1(radius1);

Circle circle2(radius2);

compareTwoCircles(circle1, circle2);

return 0;

}

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Create a program with these overloaded functions named findMax:**
   1. **One that finds maximum between two integers**
   2. **One that finds maximum between two floating-point numbers**
   3. **One that finds maximum among three integers**
   4. **One that finds maximum between an integer and a float**

#include <iostream>

using namespace std;

class Maximum

{

public:

int findMax(int a, int b)

{

return (a > b) ? a : b;

}

float findMax(float a, float b)

{

return (a > b) ? a : b;

}

int findMax(int a, int b, int c)

{

return (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c);

}

float findMax(int a, float b)

{

return (a > b) ? a : b;

}

};

int main()

{

Maximum maximum;

int int1, int2, int3;

float float1, float2;

cout << "Enter two integers: ";

cin >> int1 >> int2;

cout << "Enter two floating-point numbers: ";

cin >> float1 >> float2;

cout << "Enter three integers: ";

cin >> int1 >> int2 >> int3;

cout << "Maximum between two integers: " << maximum.findMax(int1, int2) << endl;

cout << "Maximum between two floating-point numbers: " << maximum.findMax(float1, float2) << endl;

cout << "Maximum among three integers: " << maximum.findMax(int1, int2, int3) << endl;

cout << "Maximum between an integer and a float: " << maximum.findMax(int1, float1) << endl;

return 0;

}

**OUTPUT:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Task 3: Basics of File Handling**

**Write a program that reads the titles of 10 books (use an array of 150 characters) and writes them in a binary file selected by the user. The program should read a title and display a message to indicate if it is contained in the file or not.**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <limits>

#include <stdexcept>

using namespace std;

const int MAX\_BOOKS = 10;

const int MAX\_TITLE\_LENGTH = 150;

void writeBookTitles(const string& filename, char books[][MAX\_TITLE\_LENGTH])

{

ofstream outFile(filename, ios::binary);

if (!outFile)

{

cout << "Error opening file for writing.\n";

return;

}

for (int i = 0; i < MAX\_BOOKS; ++i)

{

outFile.write(books[i], MAX\_TITLE\_LENGTH);

}

outFile.close();

}

bool searchBookTitle(const string& filename, const string& title)

{

ifstream inFile(filename, ios::binary);

if (!inFile)

{

cout << "Error opening file for reading.\n";

return false;

}

char buffer[MAX\_TITLE\_LENGTH];

while (inFile.read(buffer, MAX\_TITLE\_LENGTH))

{

if (title == buffer)

{

return true;

}

}

inFile.close();

return false;

}

struct Student

{

int roll;

string name;

float marks;

};

void readStudentsFromFile(const string& filename, vector<Student>& students)

{

ifstream inFile(filename);

if (!inFile) {

cout << "Student file not found. A new one will be created.\n";

return;

}

Student s;

while (inFile >> s.roll >> ws && getline(inFile, s.name, ',') && inFile >> s.marks) {

if (s.marks < 0 || s.marks > 100)

throw out\_of\_range("Invalid marks found in file.");

students.push\_back(s);

}

inFile.close();

}

void addStudentRecord(vector<Student>& students) {

Student s;

cout << "\nEnter new student details:\n";

cout << "Roll: ";

cin >> s.roll;

cin.ignore();

cout << "Name: ";

getline(cin, s.name);

cout << "Marks: ";

cin >> s.marks;

if (s.marks < 0 || s.marks > 100)

throw out\_of\_range("Marks must be between 0 and 100.");

students.push\_back(s);

}

void saveStudentsToFile(const string& filename, const vector<Student>& students)

{

ofstream outFile(filename);

if (!outFile)

{

cerr << "Failed to save students.\n";

return;

}

for (const auto& s : students)

{

outFile << s.roll << " " << s.name << "," << s.marks << endl;

}

outFile.close();

}

int main()

{

char books[MAX\_BOOKS][MAX\_TITLE\_LENGTH];

string bookFile;

cout << "Enter binary filename to store book titles: ";

getline(cin, bookFile);

cout << "Enter 10 book titles:\n";

for (int i = 0; i < MAX\_BOOKS; ++i)

{

cout << "Book " << i + 1 << ": ";

cin.getline(books[i], MAX\_TITLE\_LENGTH);

}

writeBookTitles(bookFile, books);

string searchTitle;

cout << "\nEnter book title to search: ";

getline(cin, searchTitle);

if (searchBookTitle(bookFile, searchTitle))

cout << "The book \"" << searchTitle << "\" is in the file.\n";

else

cout << "The book \"" << searchTitle << "\" is not in the file.\n";

vector<Student> students;

string studentFile = "students.txt";

try

{

readStudentsFromFile(studentFile, students);

}

catch (const exception& e)

{

cerr << "Exception while reading students: " << e.what() << endl;

}

char choice;

cout << "\nDo you want to add a new student record? (y/n): ";

cin >> choice;

if (choice == 'y' || choice == 'Y') {

try

{

addStudentRecord(students);

saveStudentsToFile(studentFile, students);

cout << "Student record added and saved successfully.\n";

}

catch (const exception& e) {

cerr << "Error adding student: " << e.what() << endl;

}

}

return 0;

}

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**Create a program that:**

1. **Reads student records (roll, name, marks) from a text file**
2. **Throws an exception if marks are not between 0 and 100**
3. **Allows adding new records with proper validation**
4. **Saves modified records back to file**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <stdexcept>

using namespace std;

class Student

{

private:

int roll;

string name;

int marks;

public:

Student(int r, const string& n, int m) : roll(r), name(n), marks(m) {}

int getRoll() const

{

return roll;

}

string getName() const

{

return name;

}

int getMarks() const

{

return marks;

}

static void validateMarks(int marks)

{

if (marks < 0 || marks > 100)

{

throw out\_of\_range("Marks must be between 0 and 100.");

}

}

void display() const

{

cout << "Roll Number: " << roll << ", Name: " << name << ", Marks: " << marks << endl;

}

};

class StudentManager

{

private:

vector<Student> students;

string filename;

public:

StudentManager(const string& file) : filename(file)

{

readStudentRecords();

}

void readStudentRecords()

{

ifstream file(filename);

if (!file)

{

cerr << "Error opening file for reading." << endl;

return;

}

int roll, marks;

string name;

while (file >> roll)

{

file.ignore();

getline(file, name);

file>> marks;

file.ignore();

students.push\_back(Student(roll, name, marks));

}

file.close();

}

void addStudentRecord()

{

int roll, marks;

string name;

cout << "Enter student roll number: ";

cin >> roll;

cin.ignore();

cout << "Enter student name: ";

getline(cin, name);

cout << "Enter student marks: ";

cin >> marks;

try

{

Student::validateMarks(marks);

students.push\_back(Student(roll, name, marks));

cout << "New student record added successfully!" << endl;

}

catch (const out\_of\_range& e)

{

cout << "Error: " << e.what() << endl;

}

}

void displayStudentRecords() const

{

if (students.empty())

{

cout << "No records available." << endl;

return;

}

cout << "\nStudent Records:\n";

for (const auto& student : students)

{

student.display();

}

}

void saveStudentRecords() const

{

ofstream file(filename);

if (!file)

{

cerr << "Error opening file for writing." << endl;

return;

}

for (const auto& student : students)

{

file << student.getRoll() << endl;

file << student.getName() << endl;

file << student.getMarks() << endl;

}

file.close();

}

};

int main()

{

string filename = "students.txt";

StudentManager manager(filename);

int choice;

bool running = true;

while (running)

{

cout << "\nMenu:\n";

cout << "1. Show student records\n";

cout << "2. Add new student record\n";

cout << "3. Exit\n";

cout << "Enter your choice (1-3): ";

cin >> choice;

switch (choice)

{

case 1:

manager.displayStudentRecords();

break;

case 2:

manager.addStudentRecord();

break;

case 3:

manager.saveStudentRecords();

cout << "Exiting program...\n";

running = false;

break;

default:

cout << "Invalid choice, please try again.\n";

break;

}

}

return 0;

}

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A black background with white text

AI-generated content may be incorrect.**