C++ ASSIGNMENT

Exception Handling, File Handling, Templates &

Student Record Management System

SUBMITTED BY: Rahul Kumar (85)

Section 1: Exception Handling

1. Basic Exception Handling (Division)

```
#include <iostream>
#include <stdexcept>

double divide(int a, int b) {
   if (b == 0) {
      throw std::runtime_error("Division by zero!");
   }
   return static_cast<double>(a) / b;
}

int main() {
   int num1, num2;

   std::cout << "Enter two integers: ";
   if (!(std::cin >> num1 >> num2)) {
      std::cout << "Invalid input. Please enter integers only." << std::endl;
      return 1;
   }

   try {
      double result = divide(num1, num2);
      std::cout << "Result of division: " << result << std::endl;
   } catch (const std::runtime_error& error) {
      std::cerr << "Error: " << error.what() << std::endl;
   }

   return 0;
}
```

Example Output:

Enter two integers: 10 0

Error: Division by zero!

Enter two integers: 20 5

Result of division: 4

2. Custom Exception Handling (Age Exception)

```
#include <iostream>
#include <stdexcept>
#include <string>

class AgeException : public std::exception {
public:
  const char* what() const noexcept override {
    return "Age is less than 18!";
  }
};

int main() {
  int age;

  std::cout << "Enter your age: ";
  if (!(std::cin >> age)) {
    std::cout << "Invalid input. Please enter a number." << std::endl;
    return 1;
  }
```

```
    try {
        if (age < 18) {
            throw AgeException();
        }
        std::cout << "You are eligible." << std::endl;
    } catch (const AgeException& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }

    return 0;
}
```

Example Output:

```
Enter your age: 16
Error: Age is less than 18!

Enter your age: 25
You are eligible.
```

3. Multiple Catch Blocks (Number Type)

```
#include <iostream>
#include <stdexcept>

int main() {
    int num;

    std::cout << "Enter an integer: ";
    if (!(std::cin >> num)) {
        std::cout << "Invalid input. Please enter a number." << std::endl;
        return 1;
    }
```

```
    try {
        if (num < 0) {
            throw std::invalid_argument("Number is negative!");
        } else if (num == 0) {
            throw std::runtime_error("Number is zero!");
        } else {
            std::cout << "Number is positive: " << num << std::endl;
        }
    } catch (const std::invalid_argument& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    } catch (const std::runtime_error& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    } catch (...) {
        std::cerr << "Unknown exception caught!" << std::endl;
    }

    return 0;
}
```

Example Output:

```
Enter an integer: -5
Error: Number is negative!

Enter an integer: 0
Error: Number is zero!

Enter an integer: 10
Number is positive: 10
```

4. Exception Handling in Constructors (Student)

```
#include <iostream>
#include <stdexcept>
#include <string>

class Student {
private:
  std::string name;
  int marks;

public:
  Student(std::string name, int marks) : name(name) {
    if (marks < 0 || marks > 100) {
      throw std::out_of_range("Marks are invalid!");
    }
    this->marks = marks;
  }

  void displayDetails() const {
    std::cout << "Name: " << name << ", Marks: " << marks << std::endl;
  }
};

int main() {
  try {
    Student student1("Alice", 110);
    student1.displayDetails();
  } catch (const std::out_of_range& error) {
    std::cerr << "Error: " << error.what() << std::endl;
  }
```

```
    try {
        Student student2("Bob", 85);
        student2.displayDetails();
    } catch (const std::out_of_range& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }

    return 0;
}
```

Example Output:

Error: Marks are invalid!

Name: Bob, Marks: 85

Section 2: File Handling

5. Writing to a File (Student Details)

```
#include <iostream>
#include <fstream>
#include <string>

int main() {
    std::ofstream outputFile("students.txt");

    if (!outputFile.is_open()) {
        std::cerr << "Error opening file for writing!" << std::endl;
        return 1;
    }

    std::string name;
    int rollNumber, marks;
```

```
    std::cout << "Enter student name: ";
    std::cin >> name;
    std::cout << "Enter roll number: ";
    std::cin >> rollNumber;
    std::cout << "Enter marks: ";
    std::cin >> marks;

    outputFile << name << " " << rollNumber << " " << marks << std::endl;
    outputFile.close();

    std::cout << "Student details written to file." << std::endl;

    return 0;
}
```

Example Interactions and "students.txt" Contents:

- **Input:**
    - Name: John
    - Roll Number: 101
    - Marks: 75
- **Output:** "Student details written to file."
- **Contents of students.txt:**

```
John 101 75
```

6. Reading from a File (Student Details)

```
#include <iostream>
#include <fstream>
#include <string>

int main() {
```

```
    std::ifstream inputFile("students.txt");

    if (!inputFile.is_open()) {
        std::cerr << "Error opening file for reading!" << std::endl;
        return 1;
    }

    std::string name;
    int rollNumber, marks;

    while (inputFile >> name >> rollNumber >> marks) {
        std::cout << "Name: " << name << ", Roll Number: " << rollNumber << ", Marks: " << marks << std::endl;
    }

    inputFile.close();

    return 0;
}
```

Example Output (assuming "students.txt" contains "John 101 75"):

Name: John, Roll Number: 101, Marks: 75

7. Appending Data to a File (Student Details)

```
#include <iostream>
#include <fstream>
#include <string>

int main() {
    std::ofstream outputFile("students.txt", std::ios::app);  // Open in append mode
```

```
    if (!outputFile.is_open()) {
        std::cerr << "Error opening file for appending!" << std::endl;
        return 1;
    }

    std::string name;
    int rollNumber, marks;

    std::cout << "Enter student name: ";
    std::cin >> name;
    std::cout << "Enter roll number: ";
    std::cin >> rollNumber;
    std::cout << "Enter marks: ";
    std::cin >> marks;

    outputFile << name << " " << rollNumber << " " << marks << std::endl;
    outputFile.close();

    std::cout << "Student details appended to file." << std::endl;

    return 0;
}
```

Example Interactions and "students.txt" Contents:

- **Initial students.txt contents:**

John 101 75

- **Input:**
    - Name: Jane
    - Roll Number: 102
    - Marks: 90

- **Output:** "Student details appended to file."
- **Final Contents of students.txt:**

John 101 75

Jane 102 90

8. File Copy Program

```
#include <iostream>
#include <fstream>
#include <string>

int main() {
  std::string sourceFileName, destinationFileName;

  std::cout << "Enter the source file name: ";
  std::cin >> sourceFileName;
  std::cout << "Enter the destination file name: ";
  std::cin >> destinationFileName;

  std::ifstream sourceFile(sourceFileName, std::ios::binary);
  if (!sourceFile.is_open()) {
    std::cerr << "Error opening source file!" << std::endl;
    return 1;
  }

  std::ofstream destinationFile(destinationFileName, std::ios::binary);
  if (!destinationFile.is_open()) {
    std::cerr << "Error opening destination file!" << std::endl;
    sourceFile.close(); // Close source file before exiting
    return 1;
  }
```

```
    char buffer[4096]; // Use a buffer for efficient copying
    while (sourceFile.read(buffer, sizeof(buffer))) {
        destinationFile.write(buffer, sourceFile.gcount());
    }

    destinationFile.close();
    sourceFile.close();

    std::cout << "File copied successfully." << std::endl;

    return 0;
}
```

Example Interactions:

- Assuming "source.txt" exists with some content.
- **Input:**
    - Source file name: source.txt
    - Destination file name: destination.txt
- **Output:** "File copied successfully."
- "destination.txt" will now contain the exact content of "source.txt". If source.txt doesnt exist the "Error opening source file!" message will print.

Section 3: Templates

9. Function Template (findMax)

```
#include <iostream>

template <typename T>
T findMax(T a, T b) {
    return (a > b) ? a : b;
}
```

```
int main() {
    int intMax = findMax(5, 10);
    double doubleMax = findMax(5.5, 3.2);
    char charMax = findMax('a', 'z');

    std::cout << "Max of 5 and 10: " << intMax << std::endl;
    std::cout << "Max of 5.5 and 3.2: " << doubleMax << std::endl;
    std::cout << "Max of 'a' and 'z': " << charMax << std::endl;

    return 0;
}
```

Example Output:

```
Max of 5 and 10: 10
Max of 5.5 and 3.2: 5.5
Max of 'a' and 'z': z
```

10. Class Template (Array)

```
#include <iostream>
#include <stdexcept>

template <typename T>
class Array {
private:
    T* data;
    int size;
    int capacity;

public:
    Array(int capacity) : capacity(capacity), size(0) {
```

```
    data = new T[capacity];
}

~Array() {
    delete[] data;
}

void insert(T value) {
    if (size == capacity) {
        throw std::out_of_range("Array is full!");
    }
    data[size++] = value;
}

void display() const {
    for (int i = 0; i < size; ++i) {
        std::cout << data[i] << " ";
    }
    std::cout << std::endl;
}

T findMax() const {
    if (size == 0) {
        throw std::runtime_error("Array is empty!");
    }
    T maxVal = data[0];
    for (int i = 1; i < size; ++i) {
        if (data[i] > maxVal) {
            maxVal = data[i];
        }
    }
```

```
        return maxVal;
    }
};

int main() {
    try {
        Array<int> intArray(5);
        intArray.insert(10);
        intArray.insert(5);
        intArray.insert(20);

        std::cout << "Int Array: ";
        intArray.display();
        std::cout << "Max value: " << intArray.findMax() << std::endl;
    } catch (const std::exception& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }

    try {
        Array<double> doubleArray(3);
        doubleArray.insert(3.14);
        doubleArray.insert(1.618);
         std::cout << "Double Array: ";
        doubleArray.display();
        std::cout << "Max value: " << doubleArray.findMax() << std::endl;
    } catch (const std::exception& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }

    return 0;
}
```

Example Output:

Int Array: 10 5 20

Max value: 20

Double Array: 3.14 1.618

Max value: 3.14

Section 4: Student Record Management System

```
#include <iostream>
#include <fstream>
#include <string>
#include <limits> // Required for numeric_limits
#include <vector>

// Function to clear input buffer
void clearInputBuffer() {
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

template <typename T>
class Student {
public:
    std::string name;
    int rollNo;
    T marks;

    void getData() {
        std::cout << "Enter student name: ";
        std::getline(std::cin, name); // Use getline to read names with spaces

        std::cout << "Enter roll number: ";
        while (!(std::cin >> rollNo)) {
```

```
        std::cout << "Invalid input. Enter an integer for roll number: ";
        std::cin.clear();
        clearInputBuffer();
    }
    clearInputBuffer(); // Clear the newline after reading the roll number

    std::cout << "Enter marks: ";
    while (!(std::cin >> marks)) {
        std::cout << "Invalid input. Enter a numeric value for marks: ";
        std::cin.clear();
        clearInputBuffer();
    }
    clearInputBuffer(); // Clear the newline after reading marks
  }

  void showData() const {
    std::cout << "Name: " << name << ", Roll Number: " << rollNo << ", Marks: " << marks << std::endl;
  }
};

// Function to write student data to file
template <typename T>
void writeStudentToFile(const Student<T>& student, const std::string& filename) {
  std::ofstream outputFile(filename, std::ios::app); // Append mode

  if (!outputFile.is_open()) {
    throw std::runtime_error("Error opening file for writing!");
  }

  outputFile << student.name << "," << student.rollNo << "," << student.marks << std::endl;
  outputFile.close();
```

```
}

// Function to read student data from file
template <typename T>
std::vector<Student<T>> readStudentsFromFile(const std::string& filename) {
    std::ifstream inputFile(filename);
    std::vector<Student<T>> students;

    if (!inputFile.is_open()) {
        throw std::runtime_error("Error opening file for reading!");
    }

    std::string line;
    while (std::getline(inputFile, line)) {
        Student<T> student;
        std::stringstream ss(line);
        std::string token;

        std::getline(ss, student.name, ',');

        std::getline(ss, token, ',');
        try {
            student.rollNo = std::stoi(token);
        } catch (const std::invalid_argument& e) {
            std::cerr << "Warning: Invalid roll number in file. Skipping record." << std::endl;
            continue;
        } catch (const std::out_of_range& e) {
            std::cerr << "Warning: Roll number out of range in file. Skipping record." << std::endl;
            continue;
        }
```

```
        std::getline(ss, token, ',');
        try {
            student.marks = std::stod(token); // Use stod for double
        } catch (const std::invalid_argument& e) {
            std::cerr << "Warning: Invalid marks in file. Skipping record." << std::endl;
            continue;
        } catch (const std::out_of_range& e) {
            std::cerr << "Warning: Marks out of range in file. Skipping record." << std::endl;
            continue;
        }

        students.push_back(student);
    }

    inputFile.close();
    return students;
}

// Function to search for a student by Roll Number
template <typename T>
void searchStudentByRollNo(const std::string& filename, int rollNo) {
    try {
        std::vector<Student<T>> students = readStudentsFromFile<T>(filename);
        bool found = false;
        for (const auto& student : students) {
            if (student.rollNo == rollNo) {
                std::cout << "Student found:\n";
                student.showData();
                found = true;
                break;
            }
```

```
        }
        if (!found) {
            std::cout << "Student with Roll Number " << rollNo << " not found.\n";
        }
    } catch (const std::runtime_error& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }
}

int main() {
    const std::string filename = "students.txt";
    int choice, rollNo;

    do {
        std::cout << "\nStudent Record Management System\n";
        std::cout << "1. Add Student Record\n";
        std::cout << "2. Display All Records\n";
        std::cout << "3. Search Student by Roll Number\n";
        std::cout << "0. Exit\n";
        std::cout << "Enter your choice: ";

        while (!(std::cin >> choice)) {
            std::cout << "Invalid input. Enter an integer: ";
            std::cin.clear();
            clearInputBuffer();
        }
        clearInputBuffer();

        switch (choice) {
            case 1: {
                Student<double> student;
```

```
            student.getData();
            try {
                writeStudentToFile(student, filename);
                std::cout << "Student record added successfully.\n";
            } catch (const std::runtime_error& error) {
                std::cerr << "Error: " << error.what() << std::endl;
            }
            break;
        }
        case 2: {
            try {
                std::vector<Student<double>> students = readStudentsFromFile<double>(filename);
                if (students.empty()) {
                    std::cout << "No student records found.\n";
                } else {
                    std::cout << "Student Records:\n";
                    for (const auto& student : students) {
                        student.showData();
                    }
                }
            } catch (const std::runtime_error& error) {
                std::cerr << "Error: " << error.what() << std::endl;
            }
            break;
        }
        case 3: {
            std::cout << "Enter roll number to search: ";
            while (!(std::cin >> rollNo)) {
                std::cout << "Invalid input. Enter an integer for roll number: ";
                std::cin.clear();
                clearInputBuffer();
```

```
            }
            clearInputBuffer();
            searchStudentByRollNo<double>(filename, rollNo);
            break;
        }
        case 0:
            std::cout << "Exiting program.\n";
            break;
        default:
            std::cout << "Invalid choice. Please try again.\n";
    }
  } while (choice != 0);

  return 0;
}
```

Example Interactions:

1. **Add Student Record:**
    - Enter choice: 1
    - Enter student name: Alice Smith
    - Enter roll number: 101
    - Enter marks: 85.5
    - Output: Student record added successfully.
2. **Display All Records:**
    - Enter choice: 2
    - Output:

Student Records:

Name: Alice Smith, Roll Number: 101, Marks: 85.5

3. **Search Student by Roll Number:**
    - Enter choice: 3

- Enter roll number to search: 101
- Output:

Student found:

Name: Alice Smith, Roll Number: 101, Marks: 85.5

4. **Exit:**
    - Enter choice: 0
    - Output: Exiting program.