C++ ASSIGNMENT

Basics Of C++

SUBMITTED BY: Rahul Kumar (85)

Section 1: Basic Calculations and Input/Output

1. Area and Circumference of a Circle

```
#include <iostream>
#include <iomanip> // Required for setprecision

int main() {
   double radius, area, circumference;
   const double PI = 3.14159;

   std::cout << "Enter the radius of the circle: ";
   std::cin >> radius;

   area = PI * radius * radius;
   circumference = 2 * PI * radius;

   std::cout << std::fixed << std::setprecision(2); // Set precision to 2 decimal points
   std::cout << "Area: " << area << std::endl;
   std::cout << "Circumference: " << circumference << std::endl;

   return 0;
}
```

Example Interactions:

- **Input:**
    - Radius: 5
- **Output:**

Area: 78.54

Circumference: 31.42

2. Evaluating an Expression

```
#include <iostream>
#include <cmath> // Required for pow function

int main() {
    double a, b, c, d, e, result;

    std::cout << "Enter the values of a, b, c, d, and e: ";
    std::cin >> a >> b >> c >> d >> e;

    result = pow((a + b * c - d / e), 2);

    std::cout << "Result: " << result << std::endl;

    return 0;
}
```

Example Interactions:

- **Input:**
    - a: 1
    - b: 2
    - c: 3
    - d: 4
    - e: 5
- **Output:**

Result: 43.56

Section 2: Conditional Statements

3. Prime Number Check (Nested if Statements)

```
#include <iostream>


int main() {
  int number;
  std::cout << "Enter a number: ";
  std::cin >> number;


  if (number <= 1) {
    std::cout << number << " is not a prime number." << std::endl;
  } else {
    if (number == 2) {
      std::cout << number << " is a prime number." << std::endl;
    } else {
      if (number % 2 == 0) {
        std::cout << number << " is not a prime number." << std::endl;
      } else {
        if (number == 3) {
          std::cout << number << " is a prime number." << std::endl;
        }
         else if (number % 3 == 0) {
          std::cout << number << " is not a prime number." << std::endl;
        }else{
         std::cout << number << " is a prime number." << std::endl;
        }
      }
    }
  }


  return 0;
```

}

Example Interactions:

- **Input:**
    - Number: 7
- **Output:**

7 is a prime number.

- **Input:**
    - Number: 4
- **Output:**

4 is not a prime number.

Section 3: Increment Operators

4. Post-increment vs. Pre-increment

```
#include <iostream>

int main() {
  int i = 5;

  std::cout << "Initial value of i: " << i << std::endl;

  std::cout << "Post-increment (i++): " << i++ << std::endl;
  std::cout << "Value of i after post-increment: " << i << std::endl;

  i = 5; // Reset i to 5

  std::cout << "Initial value of i: " << i << std::endl;
```

```
    std::cout << "Pre-increment (++i): " << ++i << std::endl;
    std::cout << "Value of i after pre-increment: " << i << std::endl;

    return 0;
}
```

Example Output:

```
Initial value of i: 5
Post-increment (i++): 5
Value of i after post-increment: 6
Initial value of i: 5
Pre-increment (++i): 6
Value of i after pre-increment: 6
```

Section 4: Arrays

5. Sum of Even Numbers and Product of Odd Numbers in an Array

```
#include <iostream>

int main() {
    int numbers[10];
    int sumOfEven = 0;
    int productOfOdd = 1;

    std::cout << "Enter 10 integers:" << std::endl;
    for (int i = 0; i < 10; ++i) {
        std::cin >> numbers[i];
    }

    for (int i = 0; i < 10; ++i) {
```

```
    if (numbers[i] % 2 == 0) {
        sumOfEven += numbers[i];
    } else {
        productOfOdd *= numbers[i];
    }
  }

  std::cout << "Sum of even numbers: " << sumOfEven << std::endl;
  std::cout << "Product of odd numbers: " << productOfOdd << std::endl;

  return 0;
}
```

Example Interactions:

- **Input:**
    - 1 2 3 4 5 6 7 8 9 10
- **Output:**

Sum of even numbers: 30

Product of odd numbers: 945

Section 5: Matrices

6. Transpose of a 3x3 Matrix

```
#include <iostream>

int main() {
  int matrix[3][3];

  std::cout << "Enter the elements of the 3x3 matrix:" << std::endl;
  for (int i = 0; i < 3; ++i) {
```

```
    for (int j = 0; j < 3; ++j) {
        std::cin >> matrix[i][j];
    }
  }

  std::cout << "Transpose of the matrix:" << std::endl;
  for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 3; ++j) {
        std::cout << matrix[j][i] << " ";
    }
    std::cout << std::endl;
  }

  return 0;
}
```

Example Interactions:

- **Input:**

1 2 3

4 5 6

7 8 9

- **Output:**

Transpose of the matrix:

1 4 7

2 5 8

3 6 9

Section 6: Strings

7. Counting Vowels, Consonants, Digits, and Special Characters

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

int main() {
  std::string str;
  int vowels = 0, consonants = 0, digits = 0, specialChars = 0;

  std::cout << "Enter a string: ";
  std::getline(std::cin, str); // Use getline to read the entire line

  for (char c : str) {
    if (isalpha(c)) {
      c = tolower(c); // Convert to lowercase for easy comparison
      if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {
        vowels++;
      } else {
        consonants++;
      }
    } else if (isdigit(c)) {
      digits++;
    } else {
      specialChars++;
    }
  }

  std::cout << "Vowels: " << vowels << std::endl;
  std::cout << "Consonants: " << consonants << std::endl;
  std::cout << "Digits: " << digits << std::endl;
  std::cout << "Special characters: " << specialChars << std::endl;
```

```
    return 0;
}
```

Example Interactions:

- **Input:**
    - Hello, World! 123
- **Output:**

Vowels: 3

Consonants: 7

Digits: 3

Special characters: 3

Section 7: Patterns

8. Printing a Number Pattern

```
#include <iostream>

int main() {
    int n;
    std::cout << "Enter the number of rows: ";
    std::cin >> n;

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= i; ++j) {
            std::cout << j << " ";
        }
        std::cout << std::endl;
    }
```

```
    return 0;
}
```

Example Interactions:

- **Input:**
    - Number of rows: 4
- **Output:**

```
1
1 2
1 2 3
1 2 3 4
```

Section 8: Function Pointers and Dynamic Binding

9. Dynamic Binding with Function Pointers

```
#include <iostream>

int add(int a, int b) {
    return a + b;
}

int subtract(int a, int b) {
    return a - b;
}

int main() {
    int a, b, choice;

    std::cout << "Enter two integers: ";
    std::cin >> a >> b;
```

```
    std::cout << "Enter 1 for addition, 2 for subtraction: ";
    std::cin >> choice;

    int (*operation)(int, int); // Function pointer declaration

    if (choice == 1) {
        operation = add;
    } else if (choice == 2) {
        operation = subtract;
    } else {
        std::cout << "Invalid choice." << std::endl;
        return 1;
    }

    int result = operation(a, b); // Dynamic function call
    std::cout << "Result: " << result << std::endl;

    return 0;
}
```

Example Interactions:

- **Input:**
    - Two integers: 10 5
    - Choice: 1
- **Output:**

Result: 15

- **Input:**
    - Two integers: 10 5
    - Choice: 2

- **Output:**

Result: 5

Section 9: Function Calls and Operator Input

10. Performing Operations Based on User Input

```
#include <iostream>

int performOperation(int a, int b, char op) {
  switch (op) {
    case '+':
      return a + b;
    case '-':
      return a - b;
    case '*':
      return a * b;
    case '/':
      if (b == 0) {
        std::cout << "Error: Division by zero!" << std::endl;
        return 0;
      }
      return a / b;
    default:
      std::cout << "Error: Invalid operator!" << std::endl;
      return 0;
  }
}

int main() {
  int num1, num2, result;
```

```
    char op;

    std::cout << "Enter two integers: ";
    std::cin >> num1 >> num2;
    std::cout << "Enter an operator (+, -, *, /): ";
    std::cin >> op;

    result = performOperation(num1, num2, op);
    std::cout << "Result: " << result << std::endl;

    return 0;
}
```

Example Interactions:

- **Input:**
    - Two integers: 10 5
    - Operator: +
- **Output:**

Result: 15

- **Input:**
    - Two integers: 10 5
    - Operator: /
- **Output:**

Result: 2

- **Input:**
    - Two integers: 10 0
    - Operator: /
- **Output:**

Error: Division by zero!

Result: 0

C++ ASSIGNMENT

OOPs-Introduction

SUBMITTED BY: Rahul Kumar (85)

Section 1: Classes, Objects, Constructors, Destructors

1. Understanding Classes and Objects (Student)

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

class Student {
public:
    std::string name;
    int age;
    char grade;

    void displayDetails() const {
        std::cout << "Name: " << name << std::endl;
        std::cout << "Age: " << age << std::endl;
        std::cout << "Grade: " << grade << std::endl;
    }
};

int main() {
    Student student1;
    student1.name = "Alice";
    student1.age = 20;
    student1.grade = 'A';

    student1.displayDetails();

    return 0;
}
```

Example Output:

Name: Alice

Age: 20

Grade: A

2. Constructors and Destructors (Car)

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

class Car {
public:
    std::string brand;
    std::string model;
    int year;

    Car(std::string brand, std::string model, int year) : brand(brand), model(model), year(year) {
        std::cout << "Car constructor called for " << brand << " " << model << std::endl;
    }

    ~Car() {
        std::cout << "Car destructor called for " << brand << " " << model << std::endl;
    }
};

int main() {
    Car car1("Toyota", "Camry", 2022);

    {
        Car car2("Honda", "Civic", 2023);
    } // car2 is destroyed at the end of this block
```

```
    return 0; // car1 is destroyed at the end of main()
}
```

Example Output:

Car constructor called for Toyota Camry

Car constructor called for Honda Civic

Car destructor called for Honda Civic

Car destructor called for Toyota Camry

3. Dynamic Memory Allocation (Book)

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

class Book {
public:
    std::string title;
    double price;

    Book(std::string title, double price) : title(title), price(price) {}

    void displayDetails() const {
        std::cout << "Title: " << title << std::endl;
        std::cout << "Price: " << price << std::endl;
    }
};

int main() {
    Book* bookPtr = new Book("The C++ Programming Language", 49.99);

    bookPtr->displayDetails();
```

```
    delete bookPtr;
    bookPtr = nullptr;

    return 0;
}
```

Example Output:

Title: The C++ Programming Language

Price: 49.99

10. Constructor Overloading (Person)

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

class Person {
public:
    std::string name;
    int age;

    Person() : name("Unknown"), age(0) {
        std::cout << "Default constructor called" << std::endl;
    }

    Person(std::string name) : name(name), age(0) {
        std::cout << "Constructor with name called" << std::endl;
    }

    Person(std::string name, int age) : name(name), age(age) {
```

```
        std::cout << "Constructor with name and age called" << std::endl;
    }

    void displayDetails() const {
        std::cout << "Name: " << name << std::endl;
        std::cout << "Age: " << age << std::endl;
    }
};

int main() {
    Person person1;
    person1.displayDetails();

    Person person2("Bob");
    person2.displayDetails();

    Person person3("Charlie", 30);
    person3.displayDetails();

    return 0;
}
```

Example Output:

```
Default constructor called
Name: Unknown
Age: 0
Constructor with name called
Name: Bob
Age: 0
Constructor with name and age called
Name: Charlie
```

Age: 30

Section 2: Function and Operator Overloading

4. Function Overloading (MathOperations)

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

class MathOperations {
public:
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }

    std::string add(std::string a, std::string b) {
        return a + b;
    }
};

int main() {
    MathOperations math;

    std::cout << "Sum of integers: " << math.add(5, 10) << std::endl;
    std::cout << "Sum of doubles: " << math.add(5.5, 3.2) << std::endl;
    std::cout << "Concatenation of strings: " << math.add("Hello, ", "World!") << std::endl;

    return 0;
}
```

Example Output:

Sum of integers: 15

Sum of doubles: 8.7

Concatenation of strings: Hello, World!

7. Operator Overloading (+ Operator) (Complex)

```
#include <iostream>

class Complex {
public:
    double real;
    double imaginary;

    Complex(double real = 0.0, double imaginary = 0.0) : real(real), imaginary(imaginary) {}

    Complex operator+(const Complex& other) const {
        return Complex(real + other.real, imaginary + other.imaginary);
    }

    void display() const {
        std::cout << real << " + " << imaginary << "i" << std::endl;
    }
};

int main() {
    Complex c1(1.0, 2.0);
    Complex c2(3.0, 4.0);

    Complex c3 = c1 + c2;
    c3.display();
```

```
    return 0;
}
```

Example Output:

```
4 + 6i
```

8. Operator Overloading (== Operator) (Point)

```
#include <iostream>

class Point {
public:
    int x, y;

    Point(int x = 0, int y = 0) : x(x), y(y) {}

    bool operator==(const Point& other) const {
        return (x == other.x) && (y == other.y);
    }
};

int main() {
    Point p1(1, 2);
    Point p2(1, 2);
    Point p3(3, 4);

    if (p1 == p2) {
        std::cout << "p1 and p2 are equal" << std::endl;
    } else {
        std::cout << "p1 and p2 are not equal" << std::endl;
    }
```

```
    if (p1 == p3) {
        std::cout << "p1 and p3 are equal" << std::endl;
    } else {
        std::cout << "p1 and p3 are not equal" << std::endl;
    }

    return 0;
}
```

Example Output:

p1 and p2 are equal

p1 and p3 are not equal

9. Overloading Unary ++ Operator (Counter)

```
#include <iostream>

class Counter {
private:
    int value;

public:
    Counter(int value = 0) : value(value) {}

    // Pre-increment
    Counter& operator++() {
        ++value;
        return *this;
    }
```

```
    // Post-increment
    Counter operator++(int) {
        Counter temp = *this;
        ++value;
        return temp;
    }

    int getValue() const {
        return value;
    }
};

int main() {
    Counter c1(5);

    std::cout << "Initial value: " << c1.getValue() << std::endl;

    Counter c2 = c1++; // Post-increment
    std::cout << "Post-increment value of c1: " << c1.getValue() << std::endl;
    std::cout << "Value of c2 (post-increment): " << c2.getValue() << std::endl;

    Counter c3 = ++c1; // Pre-increment
    std::cout << "Pre-increment value of c1: " << c1.getValue() << std::endl;
    std::cout << "Value of c3 (pre-increment): " << c3.getValue() << std::endl;

    return 0;
}
```

Example Output:

```
Initial value: 5
Post-increment value of c1: 6
```

Value of c2 (post-increment): 5

Pre-increment value of c1: 7

Value of c3 (pre-increment): 7

12. Operator Overloading (<< and >> for Input/Output Stream) (Time)

```
#include <iostream>

class Time {
public:
  int hours;
  int minutes;

  Time(int hours = 0, int minutes = 0) : hours(hours), minutes(minutes) {}

  friend std::ostream& operator<<(std::ostream& os, const Time& time) {
    os << time.hours << ":" << time.minutes;
    return os;
  }

  friend std::istream& operator>>(std::istream& is, Time& time) {
    std::cout << "Enter hours: ";
    is >> time.hours;
    std::cout << "Enter minutes: ";
    is >> time.minutes;
    return is;
  }
};

int main() {
  Time t1;
  std::cin >> t1; // Input time
```

```
    std::cout << "The time is: " << t1 << std::endl; // Output time

    return 0;
}
```

Example Interactions:

- **Input:**
  - Enter hours: 10
  - Enter minutes: 30
- **Output:**
  - The time is: 10:30

Section 3: Friend Functions and Pass by Value/Reference

5. Friend Function (Rectangle)

```
#include <iostream>

class Rectangle {
private:
    int length;
    int width;

public:
    Rectangle(int length = 0, int width = 0) : length(length), width(width) {}

    friend int calculateArea(const Rectangle& rect);
};

int calculateArea(const Rectangle& rect) {
    return rect.length * rect.width;
}
```

```
int main() {
    Rectangle rect(5, 10);
    int area = calculateArea(rect);
    std::cout << "Area: " << area << std::endl;

    return 0;
}
```

Example Output:

```
Area: 50
```

6. Pass by Value vs. Pass by Reference (Number)

```
#include <iostream>

class Number {
public:
    int value;

    Number(int value = 0) : value(value) {}

    void modifyValue(Number num) { // Pass by value
        num.value = 100;
        std::cout << "Inside modifyValue: " << num.value << std::endl;
    }

    void modifyReference(Number& num) { // Pass by reference
        num.value = 200;
        std::cout << "Inside modifyReference: " << num.value << std::endl;
    }
};
```

```
int main() {
    Number n(50);

    std::cout << "Original value: " << n.value << std::endl;

    n.modifyValue(n);
    std::cout << "After modifyValue: " << n.value << std::endl;

    n.modifyReference(n);
    std::cout << "After modifyReference: " << n.value << std::endl;

    return 0;
}
```

Example Output:

```
Original value: 50
Inside modifyValue: 100
After modifyValue: 50
Inside modifyReference: 200
After modifyReference: 200
```

11. Friend Function with Two Classes (ClassA and ClassB)

```
#include <iostream>

class ClassB; // Forward declaration

class ClassA {
private:
    int valueA;
```

```
public:
    ClassA(int valueA = 0) : valueA(valueA) {}

    friend int sumObjects(const ClassA& a, const ClassB& b);
};

class ClassB {
private:
    int valueB;

public:
    ClassB(int valueB = 0) : valueB(valueB) {}

    friend int sumObjects(const ClassA& a, const ClassB& b);
};

int sumObjects(const ClassA& a, const ClassB& b) {
    return a.valueA + b.valueB;
}

int main() {
    ClassA a(10);
    ClassB b(20);

    int sum = sumObjects(a, b);
    std::cout << "Sum: " << sum << std::endl;

    return 0;
}
```

Example Output:

Sum: 30

C++ ASSIGNMENT

Inheritance and Types of Inheritance

SUBMITTED BY: Rahul Kumar (85)

Assignment-Solutions

Section 1: Inheritance and Types of Inheritance

1. Single Inheritance

A. Employee-Manager

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

class Employee {
protected:
    std::string name;
    int id;
    double salary;

public:
    Employee(std::string name, int id, double salary) : name(name), id(id), salary(salary) {}

    void displayDetails() const {
        std::cout << "Name: " << name << std::endl;
        std::cout << "ID: " << id << std::endl;
        std::cout << "Salary: " << salary << std::endl;
    }

    double getSalary() const {
        return salary;
    }
};

class Manager : public Employee {
private:
    double bonus;
```

```
public:
    Manager(std::string name, int id, double salary, double bonus) : Employee(name, id, salary), bonus(bonus) {}

    double calculateTotalSalary() const {
        return getSalary() + bonus;
    }

    void displayDetails() const {
        Employee::displayDetails();
        std::cout << "Bonus: " << bonus << std::endl;
        std::cout << "Total Salary: " << calculateTotalSalary() << std::endl;
    }
};

int main() {
    Manager manager("Rahul Kumar", 101, 60000.0, 15000.0);
    manager.displayDetails();
    return 0;
}
```

Example Output:

```
Name: Rahul Kumar
ID: 101
Salary: 60000
Bonus: 15000
Total Salary: 75000
```

B. Person-Student-Teacher

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

class Person {
protected:
    std::string name;
    int age;

public:
    Person(std::string name, int age) : name(name), age(age) {}

    void displayDetails() const {
        std::cout << "Name: " << name << std::endl;
        std::cout << "Age: " << age << std::endl;
    }
};

class Student : public Person {
private:
    int studentID;
    std::string course;

public:
    Student(std::string name, int age, int studentID, std::string course) : Person(name, age), studentID(studentID), course(course) {}

    void displayDetails() const {
        Person::displayDetails();
        std::cout << "StudentID: " << studentID << std::endl;
        std::cout << "Course: " << course << std::endl;
    }
```

```
};

class Teacher : public Person {
private:
    int teacherID;
    std::string subject;

public:
    Teacher(std::string name, int age, int teacherID, std::string subject) : Person(name, age), teacherID(teacherID), subject(subject) {}

    void displayDetails() const {
        Person::displayDetails();
        std::cout << "TeacherID: " << teacherID << std::endl;
        std::cout << "Subject: " << subject << std::endl;
    }
};

int main() {
    Student student("Alice", 20, 12345, "Computer Science");
    Teacher teacher("Bob", 35, 67890, "Mathematics");

    student.displayDetails();
    std::cout << std::endl;
    teacher.displayDetails();

    return 0;
}
```

Example Output:

Name: Alice

Age: 20

StudentID: 12345

Course: Computer Science

Name: Bob

Age: 35

TeacherID: 67890

Subject: Mathematics

C. BankAccount-SavingsAccount

```
#include <iostream>
#include <string>
class BankAccount {
protected:
  std::string accountNumber;
  double balance;

public:
  BankAccount(std::string accountNumber, double initialBalance) :
accountNumber(accountNumber), balance(initialBalance) {}

  void deposit(double amount) {
    if (amount > 0) {
      balance += amount;
      std::cout << "Deposit successful. New balance: " << balance << std::endl;
    } else {
      std::cout << "Invalid deposit amount." << std::endl;
    }
  }

  void withdraw(double amount) {
```

```
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        std::cout << "Withdrawal successful. New balance: " << balance << std::endl;
    } else {
        std::cout << "Insufficient funds or invalid amount." << std::endl;
    }
  }

  double getBalance() const {
    return balance;
  }

  void displayDetails() const {
    std::cout << "Account Number: " << accountNumber << std::endl;
    std::cout << "Balance: " << balance << std::endl;
  }
};

class SavingsAccount : public BankAccount {
private:
  double interestRate;

public:
  SavingsAccount(std::string accountNumber, double initialBalance, double interestRate)
    : BankAccount(accountNumber, initialBalance), interestRate(interestRate) {}

  double calculateInterest() const {
    return getBalance() * interestRate;
  }

  void displayDetails() const {
```

```
    BankAccount::displayDetails();
    std::cout << "Interest Rate: " << interestRate << std::endl;
    std::cout << "Calculated Interest: " << calculateInterest() << std::endl;
  }
};

int main() {
  SavingsAccount savings("123456789", 1000.0, 0.05);
  savings.displayDetails();
  savings.deposit(500.0);
  savings.withdraw(200.0);
  std::cout << "Final Balance: " << savings.getBalance() << std::endl;
  return 0;
}
```

Example Output:

```
Account Number: 123456789
Balance: 1000
Interest Rate: 0.05
Calculated Interest: 50
Deposit successful. New balance: 1500
Withdrawal successful. New balance: 1300
Final Balance: 1300
```

2. Multilevel Inheritance

A. Person-Student-Graduate Student

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

class Person {
protected:
  std::string name;
```

```
    int age;

public:
    Person(std::string name, int age) : name(name), age(age) {}

    void displayDetails() const {
        std::cout << "Name: " << name << std::endl;
        std::cout << "Age: " << age << std::endl;
    }
};

class Student : public Person {
protected:
    int rollNumber;
    std::string course;

public:
    Student(std::string name, int age, int rollNumber, std::string course) : Person(name, age), rollNumber(rollNumber), course(course) {}

    void displayDetails() const {
        Person::displayDetails();
        std::cout << "Roll Number: " << rollNumber << std::endl;
        std::cout << "Course: " << course << std::endl;
    }
};

class GraduateStudent : public Student {
private:
    std::string thesisTitle;
```

```
public:
    GraduateStudent(std::string name, int age, int rollNumber, std::string course, std::string thesisTitle)
        : Student(name, age, rollNumber, course), thesisTitle(thesisTitle) {}

    void displayDetails() const {
        Student::displayDetails();
        std::cout << "Thesis Title: " << thesisTitle << std::endl;
    }
};

int main() {
    GraduateStudent gradStudent("Charlie", 25, 54321, "Computer Science", "Advanced Algorithms");
    gradStudent.displayDetails();
    return 0;
}
```

Example Output:

```
Name: Charlie
Age: 25
Roll Number: 54321
Course: Computer Science
Thesis Title: Advanced Algorithms
```

B. Animal-Mammal-Dog

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

class Animal {
public:
    void eat() {
        std::cout << "Animal is eating" << std::endl;
    }
```

```
};

class Mammal : public Animal {
public:
  void walk() {
    std::cout << "Mammal is walking" << std::endl;
  }
};

class Dog : public Mammal {
public:
  void bark() {
    std::cout << "Dog is barking" << std::endl;
  }
};

int main() {
  Dog dog;
  dog.eat();
  dog.walk();
  dog.bark();
  return 0;
}
```

Example Output:

```
Animal is eating
Mammal is walking
Dog is barking
```

3. Multiple Inheritance

A. Sports-Academics-Student Performance

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

class Sports {
protected:
    std::string sportName;
    int score;

public:
    Sports(std::string sportName, int score) : sportName(sportName), score(score) {}

    void displaySportsDetails() const {
        std::cout << "Sport: " << sportName << std::endl;
        std::cout << "Score: " << score << std::endl;
    }
};

class Academics {
protected:
    std::string subject;
    int marks;

public:
    Academics(std::string subject, int marks) : subject(subject), marks(marks) {}

    void displayAcademicsDetails() const {
        std::cout << "Subject: " << subject << std::endl;
        std::cout << "Marks: " << marks << std::endl;
    }
};
```

```
class StudentPerformance : public Sports, public Academics {
private:
  int totalPerformance;

public:
  StudentPerformance(std::string sportName, int score, std::string subject, int marks)
    : Sports(sportName, score), Academics(subject, marks), totalPerformance(score + marks) {}

  void displayDetails() const {
    displaySportsDetails();
    displayAcademicsDetails();
    std::cout << "Total Performance: " << totalPerformance << std::endl;
  }
};

int main() {
  StudentPerformance student("Football", 90, "Math", 85);
  student.displayDetails();
  return 0;
}
```

Example Output:

```
Sport: Football
Score: 90
Subject: Math
Marks: 85
Total Performance: 175
```

B. Employee-Person-Manager

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

```
class Person {
protected:
    std::string name;
    int age;

public:
    Person(std::string name, int age) : name(name), age(age) {}

    void displayPersonDetails() const {
        std::cout << "Name: " << name << std::endl;
        std::cout << "Age: " << age << std::endl;
    }
};

class Employee {
protected:
    int employeeID;
    double salary;

public:
    Employee(int employeeID, double salary) : employeeID(employeeID), salary(salary) {}

    void displayEmployeeDetails() const {
        std::cout << "Employee ID: " << employeeID << std::endl;
        std::cout << "Salary: " << salary << std::endl;
    }
};

class Manager : public Person, public Employee {
private:
```

```
    std::string department;

public:
    Manager(std::string name, int age, int employeeID, double salary, std::string department)
        : Person(name, age), Employee(employeeID, salary), department(department) {}

    void displayDetails() const {
        displayPersonDetails();
        displayEmployeeDetails();
        std::cout << "Department: " << department << std::endl;
    }
};

int main() {
    Manager manager("David", 40, 112233, 75000.0, "Sales");
    manager.displayDetails();
    return 0;
}
```

Example Output:

Name: David

Age: 40

Employee ID: 112233

Salary: 75000

Department: Sales

4. Hierarchical Inheritance

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

class Vehicle {
```

```
protected:
    std::string brand;
    int year;

public:
    Vehicle(std::string brand, int year) : brand(brand), year(year) {}

    void displayVehicleDetails() const {
        std::cout << "Brand: " << brand << std::endl;
        std::cout << "Year: " << year << std::endl;
    }
};

class Car : public Vehicle {
private:
    std::string fuelType;

public:
    Car(std::string brand, int year, std::string fuelType) : Vehicle(brand, year), fuelType(fuelType) {}

    void displayDetails() const {
        displayVehicleDetails();
        std::cout << "Fuel Type: " << fuelType << std::endl;
    }
};

class Bike : public Vehicle {
private:
    int engineCC;

public:
```

```
    Bike(std::string brand, int year, int engineCC) : Vehicle(brand, year), engineCC(engineCC) {}

    void displayDetails() const {
        displayVehicleDetails();
        std::cout << "Engine CC: " << engineCC << std::endl;
    }
};

int main() {
    Car car("Toyota", 2022, "Petrol");
    Bike bike("Honda", 2023, 150);

    car.displayDetails();
    std::cout << std::endl;
    bike.displayDetails();

    return 0;
}
```

Example Output:

Brand: Toyota

Year: 2022

Fuel Type: Petrol

Brand: Honda

Year: 2023

Engine CC: 150

5. Hybrid Inheritance

A. Vehicle-Car/Bike-Sports Car

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

class Vehicle {
protected:
    std::string brand;
    int speed;

public:
    Vehicle(std::string brand, int speed) : brand(brand), speed(speed) {}

    void displayVehicleDetails() const {
        std::cout << "Brand: " << brand << std::endl;
        std::cout << "Speed: " << speed << std::endl;
    }
};

class Car : virtual public Vehicle {
protected:
    int numDoors;

public:
    Car(std::string brand, int speed, int numDoors) : Vehicle(brand, speed), numDoors(numDoors) {}

    void displayCarDetails() const {
        displayVehicleDetails();
        std::cout << "Number of Doors: " << numDoors << std::endl;
    }
};

class Bike : virtual public Vehicle {
```

```
protected:
    bool hasGear;

public:
    Bike(std::string brand, int speed, bool hasGear) : Vehicle(brand, speed), hasGear(hasGear) {}

    void displayBikeDetails() const {
        displayVehicleDetails();
        std::cout << "Has Gear: " << (hasGear ? "Yes" : "No") << std::endl;
    }
};

class SportsCar : public Car, public Bike {
private:
    bool turbo;

public:
    SportsCar(std::string brand, int speed, int numDoors, bool hasGear, bool turbo)
        : Vehicle(brand,speed), Car(brand, speed, numDoors), Bike(brand, speed, hasGear), turbo(turbo)
{}

    void turboMode() {
        if (turbo) {
            std::cout << "Turbo mode activated!" << std::endl;
        } else {
            std::cout << "Turbo mode not available." << std::endl;
        }
    }

        void displayDetails() const {
        displayVehicleDetails();
```

```
        std::cout << "Number of Doors: " << numDoors << std::endl;
        std::cout << "Has Gear: " << (hasGear ? "Yes" : "No") << std::endl;
        std::cout << "Turbo: " << (turbo ? "Yes" : "No") << std::endl;
    }
};

int main() {
    SportsCar sportsCar("Ferrari", 250, 2, true, true);
    sportsCar.displayDetails();
    sportsCar.turboMode();
    return 0;
}
```

Example Output:

Brand: Ferrari

Speed: 250

Number of Doors: 2

Has Gear: Yes

Turbo: Yes

Turbo mode activated!

B. Person-Student/Teacher-Teaching Assistant

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

class Person {
protected:
    std::string name;
    int age;

public:
```

```
    Person(std::string name, int age) : name(name), age(age) {}

    void displayPersonDetails() const {
        std::cout << "Name: " << name << std::endl;
        std::cout << "Age: " << age << std::endl;
    }
};

class Student : virtual public Person {
protected:
    int studentID;

public:
    Student(std::string name, int age, int studentID) : Person(name, age), studentID(studentID) {}

    void displayStudentDetails() const {
        displayPersonDetails();
        std::cout << "Student ID: " << studentID << std::endl;
    }
};

class Teacher : virtual public Person {
protected:
    std::string subject;

public:
    Teacher(std::string name, int age, std::string subject) : Person(name, age), subject(subject) {}

    void displayTeacherDetails() const {
        displayPersonDetails();
        std::cout << "Subject: " << subject << std::endl;
```

```
    }
};

class TeachingAssistant : public Student, public Teacher {
private:
    std::string labSection;

public:
    TeachingAssistant(std::string name, int age, int studentID, std::string subject, std::string labSection)
        : Person(name, age), Student(name, age, studentID), Teacher(name, age, subject), labSection(labSection) {}

    void displayDetails() const {
        displayStudentDetails();
        displayTeacherDetails();
        std::cout << "Lab Section: " << labSection << std::endl;
    }
};

int main() {
    TeachingAssistant ta("Eve", 28, 98765, "Physics", "Lab A");
    ta.displayDetails();
    return 0;
}
```

Example Output:

```
Name: Eve
Age: 28
Student ID: 98765
Subject: Physics
```

Lab Section: Lab A

Section 2: Dynamic Polymorphism and Virtual Functions

6. Virtual Function for Method Overriding

```
#include <iostream>
class Shape {
public:
   virtual double area() {
      return 0.0;
   }
};

class Circle : public Shape {
private:
   double radius;

public:
   Circle(double radius) : radius(radius) {}

   double area() override {
      return 3.14159 * radius * radius;
   }
};

class Rectangle : public Shape {
private:
   double length;
   double breadth;

public:
   Rectangle(double length, double breadth) : length(length), breadth(breadth) {}
```

```
    double area() override {
        return length * breadth;
    }
};

int main() {
    Shape* shape1 = new Circle(5.0);
    Shape* shape2 = new Rectangle(4.0, 6.0);

    std::cout << "Area of Circle: " << shape1->area() << std::endl;
    std::cout << "Area of Rectangle: " << shape2->area() << std::endl;

    delete shape1;
    delete shape2;

    return 0;
}
```

Example Output:

Area of Circle: 78.5397

Area of Rectangle: 24

7. Pure Virtual Function & Abstract Class

```
#include <iostream>
class Animal {
public:
    virtual void makeSound() = 0; // Pure virtual function
};

class Dog : public Animal {
```

```
public:
    void makeSound() override {
        std::cout << "Dog barks: Woof!" << std::endl;
    }
};

class Cat : public Animal {
public:
    void makeSound() override {
        std::cout << "Cat meows: Meow!" << std::endl;
    }
};

int main() {
    Animal* animal1 = new Dog();
    Animal* animal2 = new Cat();

    animal1->makeSound();
    animal2->makeSound();

    delete animal1;
    delete animal2;

    return 0;
}
```

Example Output:

```
Dog barks: Woof!
Cat meows: Meow!
```

8. Dynamic Method Dispatch Using Virtual Functions

```
#include <iostream>

class BankAccount {
public:
  virtual double calculateInterest() {
    return 0.0;
  }
};

class SavingsAccount : public BankAccount {
private:
  double balance;
  double interestRate;

public:
  SavingsAccount(double balance, double interestRate) : balance(balance),
interestRate(interestRate) {}

  double calculateInterest() override {
    return balance * interestRate;
  }
};

class CurrentAccount : public BankAccount {
public:
  double calculateInterest() override {
    return 0.0;
  }
};

int main() {
```

```
    BankAccount* account1 = new SavingsAccount(1000.0, 0.05);
    BankAccount* account2 = new CurrentAccount();

    std::cout << "Savings Account Interest: " << account1->calculateInterest() << std::endl;
    std::cout << "Current Account Interest: " << account2->calculateInterest() << std::endl;

    delete account1;
    delete account2;

    return 0;
}
```

Example Output:

```
Savings Account Interest: 50
Current Account Interest: 0
```

9. Virtual Destructor

```
#include <iostream>
class Base {
public:
    virtual ~Base() {
        std::cout << "Base class destructor called" << std::endl;
    }
};

class Derived : public Base {
public:
    ~Derived() override {
        std::cout << "Derived class destructor called" << std::endl;
    }
};
```

```
int main() {
    Base* basePtr = new Derived();
    delete basePtr;
    return 0;
}
```

Example Output:

```
Derived class destructor called
Base class destructor called
```

10. Abstract Class with Multiple Derived Classes

```
#include <iostream>
class Employee {
public:
    virtual double calculateSalary() = 0;
};

class FullTimeEmployee : public Employee {
private:
    double monthlySalary;

public:
    FullTimeEmployee(double monthlySalary) : monthlySalary(monthlySalary) {}

    double calculateSalary() override {
        return monthlySalary;
    }
};

class PartTimeEmployee : public Employee {
private:
```

```
    double hourlyWage;
    int hoursWorked;

public:
    PartTimeEmployee(double hourlyWage, int hoursWorked) : hourlyWage(hourlyWage),
hoursWorked(hoursWorked) {}

    double calculateSalary() override {
        return hourlyWage * hoursWorked;
    }
};

int main() {
    Employee* employee1 = new FullTimeEmployee(5000.0);
    Employee* employee2 = new PartTimeEmployee(25.0, 20);

    std::cout << "Full Time Employee Salary: " << employee1->calculateSalary() << std::endl;
    std::cout << "Part Time Employee Salary: " << employee2->calculateSalary() << std::endl;

    delete employee1;
    delete employee2;

    return 0;
}
```

Example Output:

```
Full Time Employee Salary: 5000
Part Time Employee Salary: 500
```

Section 3: Exception Handling

11. Exception Handling: Division by Zero

```
#include <iostream>

double divide(int a, int b) {
   if (b == 0) {
      throw std::runtime_error("Division by zero is not allowed.");
   }
   return static_cast<double>(a) / b;
}

int main() {
   int x, y;
   std::cout << "Enter two integers: ";
   std::cin >> x >> y;

   try {
      double result = divide(x, y);
      std::cout << "Result: " << 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 is not allowed.

Enter two integers: 10 2

Result: 5

12. Exception Handling with Multiple Catch Blocks

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

int main() {
  int num;
  std::cout << "Enter an integer: ";
  std::cin >> num;

  try {
    if (num < 0) {
      throw std::invalid_argument("Negative number is not allowed.");
    } else if (num == 0) {
      throw std::logic_error("Zero is not allowed.");
    } else if (num > 1000) {
      throw std::out_of_range("Number is too large ( > 1000).");
    } else {
      std::cout << "Valid number: " << num << std::endl;
    }
  } catch (const std::invalid_argument& error) {
    std::cerr << "Invalid Argument Error: " << error.what() << std::endl;
  } catch (const std::logic_error& error) {
    std::cerr << "Logic Error: " << error.what() << std::endl;
  } catch (const std::out_of_range& error) {
    std::cerr << "Out of Range Error: " << error.what() << std::endl;
  } catch (...) {
    std::cerr << "Unknown exception caught!" << std::endl;
  }
```

```
    return 0;
}
```

Example Output:

Enter an integer: -5

Invalid Argument Error: Negative number is not allowed.

Enter an integer: 0

Logic Error: Zero is not allowed.

Enter an integer: 1001

Out of Range Error: Number is too large (> 1000).

Enter an integer: 500

Valid number: 500

13. Exception Handling in Class Methods

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

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

public:
    Student(std::string name) : name(name), marks(0) {}

    void setMarks(int m) {
        if (m < 0 || m > 100) {
            throw std::out_of_range("Marks must be between 0 and 100.");
```

```
        }
        marks = m;
    }

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

int main() {
    Student student("Rahul");

    try {
        student.setMarks(105);
        student.displayDetails();
    } catch (const std::out_of_range& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }

    try {
        student.setMarks(85);
        student.displayDetails();
    } catch (const std::out_of_range& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }
    return 0;
}
```

Example Output:

Error: Marks must be between 0 and 100.

Name: Rahul

Marks: 85

1**4. Exception Handling in Constructors**

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

class BankAccount {
private:
  std::string accountNumber;
  double balance;

public:
  BankAccount(std::string accountNumber, double initialBalance) :
accountNumber(accountNumber) {
    if (initialBalance < 0) {
      throw std::invalid_argument("Initial balance cannot be negative.");
    }
    balance = initialBalance;
  }

  void displayDetails() const {
    std::cout << "Account Number: " << accountNumber << std::endl;
    std::cout << "Balance: " << balance << std::endl;
  }
};

int main() {
  try {
    BankAccount account("12345", -100.0);
    account.displayDetails();
```

```
    } catch (const std::invalid_argument& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }

    try {
        BankAccount account("67890", 500.0);
        account.displayDetails();
    } catch (const std::invalid_argument& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }
    return 0;
}
```

Example Output:

```
Error: Initial balance cannot be negative.
Account Number: 67890
Balance: 500
```

15. User-Defined Exception Class

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

class InvalidAgeException : public std::exception {
public:
    const char* what() const noexcept override {
        return "Invalid Age: Age must be 18 or older.";
    }
};

void checkAge(int age) {
```

```
    if (age < 18) {
        throw InvalidAgeException();
    }
    std::cout << "Age is valid." << std::endl;
}

int main() {
    try {
        checkAge(15);
    } catch (const InvalidAgeException& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }

    try {
        checkAge(25);
    } catch (const InvalidAgeException& error) {
        std::cerr << "Error: " << error.what() << std::endl;
    }
    return 0;
}
```

Example Output:

Error: Invalid Age: Age must be 18 or older.

Age is valid.

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.