1. (a) Create a class for Student with register number, name, six subject marks, result, and percentage. Write the methods for reading register number, name, and six marks and calculate result and percentage and display all details. Create array of objects and access the methods.

```
#include <iostream>
#include <string>
using namespace std;

class Student {
   string registerNumber;
   string name;
   float subjectMarks[6];
   string result;
   float percentage;

public:
   void readDetails() {
      cout << "Enter register number: ";
      cin >> registerNumber;
      cout << "Enter student name: ";
      cin >> name;
      cout << "Enter marks for six subjects: " << endl;
      for (int i = 0; i < 6; i++) {
         cout << "Subject " << i + 1 << ": ";
         cin >> subjectMarks[i];
      }
   }

   void calculateResult() {
      float totalMarks = 0;
      for (int i = 0; i < 6; i++) {
         totalMarks += subjectMarks[i];
      }
      float average = totalMarks / 6;
      percentage = (average / 100) * 100;

      if (average >= 40) {
         result = "Pass";
      } else {
         result = "Fail";
      }
   }

   void displayDetails() {
      cout << "Register Number: " << registerNumber << endl;
      cout << "Name: " << name << endl;
      cout << "Subject Marks: ";
      for (int i = 0; i < 6; i++) {
         cout << subjectMarks[i] << " ";
      }
      cout << endl;
      cout << "Result: " << result << endl;
      cout << "Percentage: " << percentage << "%" << endl;
```

```
    }
};

int main() {
    int numStudents;
    cout << "Enter the number of students: ";
    cin >> numStudents;

    Student students[numStudents];

    for (int i = 0; i < numStudents; i++) {
        cout << "Student " << i + 1 << endl;
        students[i].readDetails();
        students[i].calculateResult();
    }

    for (int i = 0; i < numStudents; i++) {
        cout << "Student " << i + 1 << " details:" << endl;
        students[i].displayDetails();
        cout << "--------------------" << endl;
    }

    return 0;
}
```

1.(b) Write a program to overload the function to calculate volume of different shapes like cube, rectangle box, and cylinder.

```
#include <iostream>
#include <cmath>
using namespace std;

const double PI = 3.14159265358979323846;

class VolumeCalculator {
public:
    double calculateVolume(double sideLength) {
        return pow(sideLength, 3); // Volume of a cube
    }

    double calculateVolume(double length, double width, double height) {
        return length * width * height; // Volume of a rectangular box
    }

    double calculateVolume(double radius, double height) {
        return PI * pow(radius, 2) * height; // Volume of a cylinder
    }
};

int main() {
```

```
    VolumeCalculator calculator;

    double sideLength, length, width, height, radius;

    cout << "Enter the side length of a cube: ";
    cin >> sideLength;
    cout << "Volume of cube: " << calculator.calculateVolume(sideLength) << endl;

    cout << "Enter the length, width, and height of a rectangular box: ";
    cin >> length >> width >> height;
    cout << "Volume of rectangular box: " << calculator.calculateVolume(length, width, height) << endl;

    cout << "Enter the radius and height of a cylinder: ";
    cin >> radius >> height;
    cout << "Volume of cylinder: " << calculator.calculateVolume(radius, height) << endl;

    return 0;
}
```

2. (a) Create a class for Complex number with real part and imaginary part. Write methods for reading real part and imaginary part, addition of two Complex numbers, subtraction of two Complex numbers and display the Complex number.

```
#include <iostream>
using namespace std;

class ComplexNumber {
    double real;
    double imaginary;

public:
    void readRealPart() {
        cout << "Enter the real part: ";
        cin >> real;
    }

    void readImaginaryPart() {
        cout << "Enter the imaginary part: ";
        cin >> imaginary;
    }

    ComplexNumber add(const ComplexNumber& other) const {
        ComplexNumber sum;
        sum.real = real + other.real;
        sum.imaginary = imaginary + other.imaginary;
        return sum;
    }

    ComplexNumber subtract(const ComplexNumber& other) const {
        ComplexNumber difference;
```

```
        difference.real = real - other.real;
        difference.imaginary = imaginary - other.imaginary;
        return difference;
    }

    void display() const {
        if (imaginary < 0) {
            cout << real << " - " << -imaginary << "i" << endl;
        } else {
            cout << real << " + " << imaginary << "i" << endl;
        }
    }
};

int main() {
    ComplexNumber num1, num2;

    cout << "Enter the first complex number:" << endl;
    num1.readRealPart();
    num1.readImaginaryPart();

    cout << "Enter the second complex number:" << endl;
    num2.readRealPart();
    num2.readImaginaryPart();

    ComplexNumber sum = num1.add(num2);
    ComplexNumber difference = num1.subtract(num2);

    cout << "Sum: ";
    sum.display();

    cout << "Difference: ";
    difference.display();

    return 0;
}
```

2.(b) Write a program to calculate compound interest using function with call by value with default parameter as rate of interest with 12%.

```
#include <iostream>
#include <cmath>
using namespace std;

double calculateCompoundInterest(double principal, double time, double rate = 12.0) {
    double amount = principal * pow(1 + (rate / 100), time);
    double interest = amount - principal;
    return interest;
}

int main() {
    double principal, time;
```

```
    cout << "Enter the principal amount: ";
    cin >> principal;
    cout << "Enter the time (in years): ";
    cin >> time;

    double interest = calculateCompoundInterest(principal, time);

    cout << "Compound interest: " << interest << endl;

    return 0;
}
```

3.(a) Create a class for String with name and length. Write the methods to read string, concatenate the string, reverse the string and display the string. Create the string objects and accesses all the methods.

```
#include <iostream>
#include <string>
using namespace std;

class String {
    string name;
    int length;

public:
    void readString() {
        cout << "Enter a string: ";
        getline(cin, name);
        length = name.length();
    }

    void concatenateString(const String& other) {
        name += other.name;
        length = name.length();
    }

    void reverseString() {
        string reversed;
        for (int i = length - 1; i >= 0; i--) {
            reversed += name[i];
        }
        name = reversed;
    }

    void displayString() const {
        cout << "String: " << name << endl;
        cout << "Length: " << length << endl;
    }
};

int main() {
    String string1, string2;
```

```
    cout << "Enter the first string:" << endl;
    string1.readString();

    cout << "Enter the second string:" << endl;
    string2.readString();

    string1.concatenateString(string2);

    cout << "Concatenated string:" << endl;
    string1.displayString();

    cout << "Reversed string:" << endl;
    string1.reverseString();
    string1.displayString();

    return 0;
}
```

3.(b) Write a program to interchange the values using function with call by reference.

```
#include <iostream>
using namespace std;

void interchangeValues(int& num1, int& num2) {
    int temp = num1;
    num1 = num2;
    num2 = temp;
}

int main() {
    int a, b;

    cout << "Enter the value of a: ";
    cin >> a;
    cout << "Enter the value of b: ";
    cin >> b;

    cout << "Before interchange: a = " << a << ", b = " << b << endl;

    interchangeValues(a, b);

    cout << "After interchange: a = " << a << ", b = " << b << endl;

    return 0;
}
```

4.(a) Create a class for Student with register number, name, six subject grades, and CGPA. Write the methods for reading register number, name, and six subject grades and calculate CGPA and display all details. Create array of objects and access the methods.

```
#include <iostream>
#include <string>
using namespace std;

const int NUM_SUBJECTS = 6;

class Student {
   int regNumber;
   string name;
   int grades[NUM_SUBJECTS];
   double cgpa;

public:
   void readRegNumber() {
      cout << "Enter the register number: ";
      cin >> regNumber;
   }

   void readName() {
      cout << "Enter the name: ";
      cin.ignore();
      getline(cin, name);
   }

   void readGrades() {
      cout << "Enter the grades for " << NUM_SUBJECTS << " subjects:" << endl;
      for (int i = 0; i < NUM_SUBJECTS; i++) {
         cout << "Subject " << i + 1 << ": ";
         cin >> grades[i];
      }
   }

   void calculateCGPA() {
      int totalMarks = 0;
      for (int i = 0; i < NUM_SUBJECTS; i++) {
         totalMarks += grades[i];
      }
      cgpa = totalMarks / static_cast<double>(NUM_SUBJECTS);
   }

   void displayDetails() const {
      cout << "Register Number: " << regNumber << endl;
      cout << "Name: " << name << endl;
      cout << "Grades:" << endl;
      for (int i = 0; i < NUM_SUBJECTS; i++) {
         cout << "Subject " << i + 1 << ": " << grades[i] << endl;
      }
      cout << "CGPA: " << cgpa << endl;
```

```
    }
};

int main() {
    const int NUM_STUDENTS = 3;
    Student students[NUM_STUDENTS];

    for (int i = 0; i < NUM_STUDENTS; i++) {
        cout << "Enter details for Student " << i + 1 << ":" << endl;
        students[i].readRegNumber();
        students[i].readName();
        students[i].readGrades();
        students[i].calculateCGPA();
    }

    for (int i = 0; i < NUM_STUDENTS; i++) {
        cout << "Details for Student " << i + 1 << ":" << endl;
        students[i].displayDetails();
        cout << endl;
    }

    return 0;
}
```

4.(b) Write a program to calculate factorial N using function with call by value.

```
#include <iostream>
using namespace std;

unsigned long long calculateFactorial(unsigned int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        unsigned long long factorial = 1;
        for (unsigned int i = 2; i <= n; i++) {
            factorial *= i;
        }
        return factorial;
    }
}

int main() {
    unsigned int number;

    cout << "Enter a non-negative integer: ";
    cin >> number;

    unsigned long long factorial = calculateFactorial(number);

    cout << "Factorial of " << number << " = " << factorial << endl;
```

```
    return 0;
}
```

5.(a) Create class for Item with code, name, and price. Write constructors (default, parameterized, and copy), display the details and static members to count the number of items created and display the count value.

```
#include <iostream>
#include <string>
using namespace std;

class Item {
    int code;
    string name;
    double price;
    static int count;

public:
    // Default constructor
    Item() : code(0), name(""), price(0.0) {
        count++;
    }

    // Parameterized constructor
    Item(int _code, const string& _name, double _price)
        : code(_code), name(_name), price(_price) {
        count++;
    }

    // Copy constructor
    Item(const Item& other)
        : code(other.code), name(other.name), price(other.price) {
        count++;
    }

    // Display item details
    void displayDetails() const {
        cout << "Code: " << code << endl;
        cout << "Name: " << name << endl;
        cout << "Price: " << price << endl;
    }

    // Static member function to display the count value
    static void displayCount() {
        cout << "Number of items created: " << count << endl;
    }
};

// Initializing the static member count
int Item::count = 0;

int main() {
```

```
    Item item1; // Default constructor
    Item item2(1, "Book", 9.99); // Parameterized constructor
    Item item3 = item2; // Copy constructor

    item1.displayDetails();
    cout << endl;
    item2.displayDetails();
    cout << endl;
    item3.displayDetails();
    cout << endl;

    Item::displayCount(); // Accessing the static member function

    return 0;
}
```

5.(b) Write a program to swap the values using function with call by address.

```
#include <iostream>
using namespace std;

void swapValues(int* num1, int* num2) {
   int temp = *num1;
   *num1 = *num2;
   *num2 = temp;
}

int main() {
   int a, b;

   cout << "Enter the value of a: ";
   cin >> a;
   cout << "Enter the value of b: ";
   cin >> b;

   cout << "Before swap: a = " << a << ", b = " << b << endl;

   swapValues(&a, &b);

   cout << "After swap: a = " << a << ", b = " << b << endl;

   return 0;
}
```

6.(a) Create a class for Complex number with real part and imaginary part. Write methods for reading real part and imaginary part, display the Complex number and over load the operator + and – using member function.

```
#include <iostream>
using namespace std;

class ComplexNumber {
   double real;
   double imaginary;

public:
   // Constructor
   ComplexNumber(double _real = 0.0, double _imaginary = 0.0)
      : real(_real), imaginary(_imaginary) {}

   // Method to read the real part
   void readRealPart() {
      cout << "Enter the real part: ";
      cin >> real;
   }

   // Method to read the imaginary part
   void readImaginaryPart() {
      cout << "Enter the imaginary part: ";
      cin >> imaginary;
   }

   // Method to display the complex number
   void displayComplexNumber() const {
      cout << real << " + " << imaginary << "i" << endl;
   }

   // Overloaded + operator
   ComplexNumber operator+(const ComplexNumber& other) const {
      ComplexNumber result;
      result.real = real + other.real;
      result.imaginary = imaginary + other.imaginary;
      return result;
   }

   // Overloaded - operator
   ComplexNumber operator-(const ComplexNumber& other) const {
      ComplexNumber result;
      result.real = real - other.real;
      result.imaginary = imaginary - other.imaginary;
      return result;
   }
};

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

```
    cout << "Enter details for Complex Number 1:" << endl;
    num1.readRealPart();
    num1.readImaginaryPart();

    cout << "Enter details for Complex Number 2:" << endl;
    num2.readRealPart();
    num2.readImaginaryPart();

    cout << "Complex Number 1: ";
    num1.displayComplexNumber();

    cout << "Complex Number 2: ";
    num2.displayComplexNumber();

    ComplexNumber sum = num1 + num2;
    cout << "Sum: ";
    sum.displayComplexNumber();

    ComplexNumber diff = num1 - num2;
    cout << "Difference: ";
    diff.displayComplexNumber();

    return 0;
}
```

6.(b) Write a program to calculate the area of triangle using inline function.

```
#include <iostream>
using namespace std;

inline float calculateTriangleArea(float base, float height) {
    return 0.5 * base * height;
}

int main() {
    float base, height;

    cout << "Enter the base of the triangle: ";
    cin >> base;

    cout << "Enter the height of the triangle: ";
    cin >> height;

    float area = calculateTriangleArea(base, height);

    cout << "The area of the triangle is: " << area << endl;

    return 0;
}
```

7.(a) Create a class for Date with date, month and year. Write constructors and over load the operator == and != using friend function.

```
#include <iostream>
using namespace std;

class Date {
   int day;
   int month;
   int year;

public:
   // Constructor
   Date(int _day = 0, int _month = 0, int _year = 0)
      : day(_day), month(_month), year(_year) {}

   // Friend function to overload the == operator
   friend bool operator==(const Date& date1, const Date& date2);

   // Friend function to overload the != operator
   friend bool operator!=(const Date& date1, const Date& date2);
};

// Overloaded == operator
bool operator==(const Date& date1, const Date& date2) {
   return (date1.day == date2.day &&
           date1.month == date2.month &&
           date1.year == date2.year);
}

// Overloaded != operator
bool operator!=(const Date& date1, const Date& date2) {
   return !(date1 == date2);
}

int main() {
   Date date1(20, 5, 2023);
   Date date2(15, 8, 2021);
   Date date3(20, 5, 2023);

   if (date1 == date2) {
      cout << "date1 is equal to date2" << endl;
   } else {
      cout << "date1 is not equal to date2" << endl;
   }

   if (date1 != date3) {
      cout << "date1 is not equal to date3" << endl;
   } else {
      cout << "date1 is equal to date3" << endl;
   }

   return 0;
```

```
}
```

7.(b)Write a program to check the number is prime number or not using function with call by value.

```
#include <iostream>
using namespace std;

bool isPrime(int number) {
    if (number <= 1) {
        return false;
    }

    for (int i = 2; i * i <= number; i++) {
        if (number % i == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int number;

    cout << "Enter a number: ";
    cin >> number;

    if (isPrime(number)) {
        cout << number << " is a prime number" << endl;
    } else {
        cout << number << " is not a prime number" << endl;
    }

    return 0;
}
```

8.(a) Create a class for String with constructor and over load the operator < and > using member function.

```
#include <iostream>
#include <string>
using namespace std;

class String {
    string str;

public:
    // Constructor
    String(const string& _str) : str(_str) {}

    // Overloaded < operator
    bool operator<(const String& other) const {
```

```
        return str < other.str;
    }

    // Overloaded > operator
    bool operator>(const String& other) const {
        return str > other.str;
    }
};

int main() {
    String str1("Hello");
    String str2("World");

    if (str1 < str2) {
        cout << "str1 is less than str2" << endl;
    } else {
        cout << "str1 is not less than str2" << endl;
    }

    if (str1 > str2) {
        cout << "str1 is greater than str2" << endl;
    } else {
        cout << "str1 is not greater than str2" << endl;
    }

    return 0;
}
```

8.(b) Write a program to check the number is Armstrong number or not using function with call by value.

```
#include <iostream>
using namespace std;

int calculatePower(int base, int exponent) {
    int result = 1;

    while (exponent > 0) {
        result *= base;
        exponent--;
    }

    return result;
}

int countDigits(int number) {
    int count = 0;

    while (number > 0) {
        number /= 10;
        count++;
    }
```

```
    return count;
}

bool isArmstrong(int number) {
    int originalNumber = number;
    int numDigits = countDigits(number);
    int sum = 0;

    while (number > 0) {
        int digit = number % 10;
        sum += calculatePower(digit, numDigits);
        number /= 10;
    }

    return (sum == originalNumber);
}

int main() {
    int number;

    cout << "Enter a number: ";
    cin >> number;

    if (isArmstrong(number)) {
        cout << number << " is an Armstrong number" << endl;
    } else {
        cout << number << " is not an Armstrong number" << endl;
    }

    return 0;
}
```

9.(a) Create a class for String with constructor and over load the operator +, == and != using friend function.

```
#include <iostream>
#include <string>
using namespace std;

class String {
    string str;

public:
    // Constructor
    String(const string& _str) : str(_str) {}

    // Friend function to overload the + operator
    friend String operator+(const String& string1, const String& string2);
```

```
    // Friend function to overload the == operator
    friend bool operator==(const String& string1, const String& string2);

    // Friend function to overload the != operator
    friend bool operator!=(const String& string1, const String& string2);

    // Function to display the string
    void display() {
        cout << str << endl;
    }
};

// Overloaded + operator
String operator+(const String& string1, const String& string2) {
    string result = string1.str + string2.str;
    return String(result);
}

// Overloaded == operator
bool operator==(const String& string1, const String& string2) {
    return string1.str == string2.str;
}

// Overloaded != operator
bool operator!=(const String& string1, const String& string2) {
    return !(string1 == string2);
}

int main() {
    String str1("Hello");
    String str2("World");

    String str3 = str1 + str2;

    str3.display();

    if (str1 == str2) {
        cout << "str1 is equal to str2" << endl;
    } else {
        cout << "str1 is not equal to str2" << endl;
    }

    if (str1 != str2) {
        cout << "str1 is not equal to str2" << endl;
    } else {
        cout << "str1 is equal to str2" << endl;
    }

    return 0;
}
```

9.(b) Write a program to calculate the area of circle using inline function.

```
#include <iostream>
using namespace std;

inline double calculateArea(double radius) {
   return 3.14159 * radius * radius;
}

int main() {
   double radius;

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

   double area = calculateArea(radius);

   cout << "The area of the circle is: " << area << endl;

   return 0;
}
```

10.(a) Create a class for Time with second, minute and hour. Write the methods to coveter type conversion from int to Time and Time to int.

```
#include <iostream>
using namespace std;

class Time {
   int seconds;
   int minutes;
   int hours;

public:
   // Constructor
   Time(int s = 0, int m = 0, int h = 0)
      : seconds(s), minutes(m), hours(h) {}

   // Convert int to Time
   void convertFromInt(int totalSeconds) {
      hours = totalSeconds / 3600;
      minutes = (totalSeconds % 3600) / 60;
      seconds = (totalSeconds % 3600) % 60;
   }

   // Convert Time to int
   int convertToInt() {
      return (hours * 3600) + (minutes * 60) + seconds;
   }
```

```
    // Display the time
    void displayTime() {
        cout << "Time: " << hours << "h " << minutes << "m " << seconds << "s" << endl;
    }
};

int main() {
    Time t1;
    int totalSeconds;

    cout << "Enter the total seconds: ";
    cin >> totalSeconds;

    // Convert int to Time
    t1.convertFromInt(totalSeconds);
    t1.displayTime();

    // Convert Time to int
    int convertedSeconds = t1.convertToInt();
    cout << "Converted to seconds: " << convertedSeconds << "s" << endl;

    return 0;
}
```

10.(b) Create a class student with register number and name and inherit the student class into personal with father name, and address. Write methods for reading and displaying the details. Create the object and accesses the methods.

```
#include <iostream>
#include <string>
using namespace std;

// Base class: Student
class Student {
protected:
    int registerNumber;
    string name;

public:
    // Constructor
    Student(int regNum, const string& studentName)
        : registerNumber(regNum), name(studentName) {}

    // Methods to read and display student details
    void readDetails() {
        cout << "Enter Register Number: ";
        cin >> registerNumber;
        cout << "Enter Name: ";
        cin.ignore();
        getline(cin, name);
```

```
    }

    void displayDetails() {
        cout << "Register Number: " << registerNumber << endl;
        cout << "Name: " << name << endl;
    }
};

// Derived class: Personal, inherits from Student
class Personal : public Student {
private:
    string fatherName;
    string address;

public:
    // Constructor
    Personal(int regNum, const string& studentName, const string& fName, const string& addr)
        : Student(regNum, studentName), fatherName(fName), address(addr) {}

    // Method to read personal details
    void readPersonalDetails() {
        readDetails();
        cout << "Enter Father's Name: ";
        getline(cin, fatherName);
        cout << "Enter Address: ";
        getline(cin, address);
    }

    // Method to display personal details
    void displayPersonalDetails() {
        displayDetails();
        cout << "Father's Name: " << fatherName << endl;
        cout << "Address: " << address << endl;
    }
};

int main() {
    Personal personal(0, "", "", "");

    personal.readPersonalDetails();
    cout << "\nPersonal Details:" << endl;
    personal.displayPersonalDetails();

    return 0;
}
```

11.(a) Create the classes for distance in m and cm and feet and inch. Write the type conversion methods to convert distance from m and cm to feet and inch feet and inch into m and cm.

```
#include <iostream>
```

```
class DistanceMCM {
private:
    int meters;
    int centimeters;

public:
    DistanceMCM(int m, int cm) : meters(m), centimeters(cm) {}

    int getMeters() const {
        return meters;
    }

    int getCentimeters() const {
        return centimeters;
    }

    void setMeters(int m) {
        meters = m;
    }

    void setCentimeters(int cm) {
        centimeters = cm;
    }

    void display() const {
        std::cout << "Distance: " << meters << " meters " << centimeters << " centimeters" << std::endl;
    }

    // Conversion methods
    DistanceMCM toFeetInch() const {
        int totalInches = (meters * 100 + centimeters) / 2.54;
        int feet = totalInches / 12;
        int inches = totalInches % 12;
        return DistanceMCM(feet, inches);
    }
};

class DistanceFTIN {
private:
    int feet;
    int inches;

public:
    DistanceFTIN(int ft, int in) : feet(ft), inches(in) {}

    int getFeet() const {
        return feet;
    }

    int getInches() const {
        return inches;
    }
```

```
    void setFeet(int ft) {
        feet = ft;
    }

    void setInches(int in) {
        inches = in;
    }

    void display() const {
        std::cout << "Distance: " << feet << " feet " << inches << " inches" << std::endl;
    }

    // Conversion methods
    DistanceFTIN toMetersCm() const {
        int totalInches = feet * 12 + inches;
        int centimeters = totalInches * 2.54;
        int meters = centimeters / 100;
        centimeters = centimeters % 100;
        return DistanceFTIN(meters, centimeters);
    }
};

int main() {
    DistanceMCM distance1(5, 20);
    distance1.display();

    DistanceFTIN distance2 = distance1.toFeetInch();
    distance2.display();

    DistanceFTIN distance3(6, 8);
    distance3.display();

    DistanceMCM distance4 = distance3.toMetersCm();
    distance4.display();

    return 0;
}
```

11.(b) Write inline function to calculate are of rectangle.

```
#include <iostream>

inline int calculateRectangleArea(int length, int width) {
    return length * width;
}

int main() {
    int length, width;
    std::cout << "Enter the length of the rectangle: ";
    std::cin >> length;
    std::cout << "Enter the width of the rectangle: ";
```

```
    std::cin >> width;

    int area = calculateRectangleArea(length, width);

    std::cout << "The area of the rectangle is: " << area << std::endl;

    return 0;
}
```

12.(a) Write a program to implement virtual function to calculate volume of different shapes.

```
#include <iostream>
using namespace std;

// Abstract base class
class Shape {
public:
  virtual double calculateVolume() = 0; // Pure virtual function
};

// Derived class for a sphere
class Sphere : public Shape {
private:
  double radius;

public:
  Sphere(double r) {
    radius = r;
  }

  double calculateVolume() {
    return (4.0 / 3.0) * 3.1415 * radius * radius * radius;
  }
};

// Derived class for a cube
class Cube : public Shape {
private:
  double side;

public:
  Cube(double s) {
    side = s;
  }

  double calculateVolume() {
    return side * side * side;
  }
};

// Derived class for a cylinder
```

```
class Cylinder : public Shape {
private:
  double radius;
  double height;

public:
  Cylinder(double r, double h) {
    radius = r;
    height = h;
  }

  double calculateVolume() {
    return 3.1415 * radius * radius * height;
  }
};

int main() {
  // Creating objects of different shapes
  Shape* shape1 = new Sphere(3.0);
  Shape* shape2 = new Cube(4.0);
  Shape* shape3 = new Cylinder(2.0, 5.0);

  // Calculating and displaying the volumes
  cout << "Volume of sphere: " << shape1->calculateVolume() << endl;
  cout << "Volume of cube: " << shape2->calculateVolume() << endl;
  cout << "Volume of cylinder: " << shape3->calculateVolume() << endl;

  // Cleaning up the objects
  delete shape1;
  delete shape2;
  delete shape3;

  return 0;
}
```

12.(b) Write a program to create an exception for marks (<0 and>100) out of range and catch the exception.

```
#include <iostream>
using namespace std;

// Custom exception class for marks out of range
class MarksOutOfRangeException : public exception {
public:
  const char* what() const throw() {
    return "Marks out of range!";
  }
};

// Function to validate marks
void validateMarks(int marks) {
  if (marks < 0 || marks > 100) {
```

```
    throw MarksOutOfRangeException();
  }
}

int main() {
  int marks;

  cout << "Enter marks: ";
  cin >> marks;

  try {
    validateMarks(marks);
    cout << "Marks are within the range." << endl;
  } catch (MarksOutOfRangeException& e) {
    cout << "Exception: " << e.what() << endl;
  }

  return 0;
}
```

13.(a)Write a program to implement virtual function to calculate area of different shapes.

```
#include <iostream>
using namespace std;

// Abstract base class
class Shape {
public:
  virtual double calculateArea() = 0; // Pure virtual function
};

// Derived class for a rectangle
class Rectangle : public Shape {
private:
  double length;
  double width;

public:
  Rectangle(double l, double w) {
    length = l;
    width = w;
  }

  double calculateArea() {
    return length * width;
  }
};

// Derived class for a circle
class Circle : public Shape {
private:
```

```
  double radius;

public:
  Circle(double r) {
    radius = r;
  }

  double calculateArea() {
    return 3.1415 * radius * radius;
  }
};

// Derived class for a triangle
class Triangle : public Shape {
private:
  double base;
  double height;

public:
  Triangle(double b, double h) {
    base = b;
    height = h;
  }

  double calculateArea() {
    return 0.5 * base * height;
  }
};

int main() {
  // Creating objects of different shapes
  Shape* shape1 = new Rectangle(4.0, 5.0);
  Shape* shape2 = new Circle(3.0);
  Shape* shape3 = new Triangle(6.0, 8.0);

  // Calculating and displaying the areas
  cout << "Area of rectangle: " << shape1->calculateArea() << endl;
  cout << "Area of circle: " << shape2->calculateArea() << endl;
  cout << "Area of triangle: " << shape3->calculateArea() << endl;

  // Cleaning up the objects
  delete shape1;
  delete shape2;
  delete shape3;

  return 0;
}
```

13.(b) Write a program to create an exception for age (<18 and>21) out of range and catch the exception.

```
#include <iostream>
using namespace std;
```

```
// Custom exception class for age out of range
class AgeOutOfRangeException : public exception {
public:
  const char* what() const throw() {
    return "Age out of range!";
  }
};

// Function to validate age
void validateAge(int age) {
  if (age < 18 || age > 21) {
    throw AgeOutOfRangeException();
  }
}

int main() {
  int age;

  cout << "Enter age: ";
  cin >> age;

  try {
    validateAge(age);
    cout << "Age is within the range." << endl;
  } catch (AgeOutOfRangeException& e) {
    cout << "Exception: " << e.what() << endl;
  }

  return 0;
}
```

14.(a) Create a Student class with id, name, and percentage. Write the methods to over load the operator >> and << using friend function.

```
#include <iostream>
using namespace std;

class Student {
private:
  int id;
  string name;
  float percentage;

public:
  // Default constructor
  Student() {}

  // Parameterized constructor
```

```
  Student(int i, string n, float p) {
    id = i;
    name = n;
    percentage = p;
  }

  // Friend function to overload the >> operator for input
  friend istream& operator>>(istream& in, Student& s) {
    cout << "Enter ID: ";
    in >> s.id;

    cout << "Enter Name: ";
    in >> s.name;

    cout << "Enter Percentage: ";
    in >> s.percentage;

    return in;
  }

  // Friend function to overload the << operator for output
  friend ostream& operator<<(ostream& out, const Student& s) {
    out << "ID: " << s.id << endl;
    out << "Name: " << s.name << endl;
    out << "Percentage: " << s.percentage << endl;

    return out;
  }
};

int main() {
  Student s;

  cout << "Enter student details:" << endl;
  cin >> s;

  cout << "Student details:" << endl;
  cout << s;

  return 0;
}
```

14.(b) Write a class template for Array to read, display and sort the values.

```
#include <iostream>
#include <algorithm>
using namespace std;
```

```
template <typename T, int SIZE>
class Array {
private:
  T data[SIZE];

public:
  // Method to read values into the array
  void readValues() {
    cout << "Enter " << SIZE << " values: ";
    for (int i = 0; i < SIZE; i++) {
      cin >> data[i];
    }
  }

  // Method to display the values in the array
  void displayValues() {
    cout << "Values in the array: ";
    for (int i = 0; i < SIZE; i++) {
      cout << data[i] << " ";
    }
    cout << endl;
  }

  // Method to sort the values in the array
  void sortValues() {
    sort(data, data + SIZE);
  }
};

int main() {
  // Create an Array object of type int with size 5
  Array<int, 5> intArray;

  // Read values into the array
  intArray.readValues();

  // Display the values
  intArray.displayValues();

  // Sort the values
  intArray.sortValues();

  // Display the sorted values
  intArray.displayValues();

  return 0;
}
```

15.(a) Create a class for Complex number with real part and imaginary part. Write methods for reading real part and imaginary part, display the Complex number and over load the operator + using member function.

```
#include <iostream>
using namespace std;

class Complex {
private:
  double real;
  double imaginary;

public:
  // Method to read the real and imaginary parts
  void read() {
    cout << "Enter the real part: ";
    cin >> real;

    cout << "Enter the imaginary part: ";
    cin >> imaginary;
  }

  // Method to display the complex number
  void display() {
    cout << "Complex number: " << real << " + " << imaginary << "i" << endl;
  }

  // Overloading the + operator using a member function
  Complex operator+(const Complex& other) {
    Complex sum;
    sum.real = real + other.real;
    sum.imaginary = imaginary + other.imaginary;
    return sum;
  }
};

int main() {
  Complex c1, c2, sum;

  cout << "Enter the first complex number:" << endl;
  c1.read();

  cout << "Enter the second complex number:" << endl;
  c2.read();

  sum = c1 + c2;

  cout << "Sum of the complex numbers:" << endl;
  sum.display();

  return 0;
}
```

15.(b) Write a function template to add three values (including complex number).

```
#include <iostream>
using namespace std;

// Template function to add three values
template <typename T>
T addThreeValues(T a, T b, T c) {
  return a + b + c;
}

int main() {
  int num1 = 10, num2 = 20, num3 = 30;
  double double1 = 1.5, double2 = 2.5, double3 = 3.5;
  Complex complex1, complex2, complex3;

  // Add three integers
  int sumIntegers = addThreeValues(num1, num2, num3);
  cout << "Sum of integers: " << sumIntegers << endl;

  // Add three doubles
  double sumDoubles = addThreeValues(double1, double2, double3);
  cout << "Sum of doubles: " << sumDoubles << endl;

  // Add three complex numbers
  Complex sumComplex;
  sumComplex = addThreeValues(complex1, complex2, complex3);
  cout << "Sum of complex numbers: ";
  sumComplex.display();

  return 0;
}
```

16.(a) Create a Student class with id, name, and percentage. Write the methods to over load the operator <or > using member function.

```
#include <iostream>
using namespace std;

class Student {
private:
  int id;
  string name;
  float percentage;

public:
  // Constructor
  Student(int i, string n, float p) {
    id = i;
```

```
    name = n;
    percentage = p;
  }

  // Overloading the < operator
  bool operator<(const Student& other) {
    return percentage < other.percentage;
  }

  // Overloading the > operator
  bool operator>(const Student& other) {
    return percentage > other.percentage;
  }

  // Method to display student details
  void display() {
    cout << "ID: " << id << endl;
    cout << "Name: " << name << endl;
    cout << "Percentage: " << percentage << endl;
  }
};

int main() {
  Student s1(1, "John", 75.5);
  Student s2(2, "Alice", 82.0);

  cout << "Student 1 details:" << endl;
  s1.display();

  cout << "Student 2 details:" << endl;
  s2.display();

  if (s1 < s2) {
    cout << "Student 1 has a lower percentage than Student 2." << endl;
  } else if (s1 > s2) {
    cout << "Student 1 has a higher percentage than Student 2." << endl;
  } else {
    cout << "Student 1 and Student 2 have the same percentage." << endl;
  }

  return 0;
}
```

16.(b) Write a function template to sort the values (including student).

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class Student {
```

```
private:
  int id;
  string name;
  float percentage;

public:
  // Constructor
  Student(int i, string n, float p) {
    id = i;
    name = n;
    percentage = p;
  }

  // Getter for percentage
  float getPercentage() const {
    return percentage;
  }

  // Method to display student details
  void display() const {
    cout << "ID: " << id << endl;
    cout << "Name: " << name << endl;
    cout << "Percentage: " << percentage << endl;
  }
};

// Function template to sort values
template <typename T>
void sortValues(vector<T>& values) {
  sort(values.begin(), values.end());
}

int main() {
  // Create a vector of integers
  vector<int> numbers = {3, 1, 4, 2, 5};
  cout << "Before sorting integers: ";
  for (const auto& num : numbers) {
    cout << num << " ";
  }
  cout << endl;

  // Sort the integers
  sortValues(numbers);
  cout << "After sorting integers: ";
  for (const auto& num : numbers) {
    cout << num << " ";
  }
  cout << endl;

  // Create a vector of Student objects
```

```
  vector<Student> students = {
    Student(1, "John", 75.5),
    Student(2, "Alice", 82.0),
    Student(3, "Bob", 90.25)
  };
  cout << "\nBefore sorting students:" << endl;
  for (const auto& student : students) {
    student.display();
    cout << endl;
  }

  // Sort the students based on percentage
  sortValues(students);
  cout << "After sorting students:" << endl;
  for (const auto& student : students) {
    student.display();
    cout << endl;
  }

  return 0;
}
```

17.(a) Create a class Account with number, name, and balance and inherit the class for Transaction with date, CR/DB type, amount and Load with date, interest. Write methods for reading and displaying and calculation Create object for Transaction and Loan classes and access the methods.

```
#include <iostream>
#include <string>
using namespace std;

class Account {
protected:
  int number;
  string name;
  double balance;

public:
  // Constructor
  Account(int num, const string& nm, double bal) {
    number = num;
    name = nm;
    balance = bal;
  }

  // Method to read account details
  virtual void read() {
    cout << "Enter account number: ";
    cin >> number;
```

```
    cout << "Enter account holder's name: ";
    cin.ignore();
    getline(cin, name);

    cout << "Enter account balance: ";
    cin >> balance;
  }

  // Method to display account details
  virtual void display() {
    cout << "Account number: " << number << endl;
    cout << "Account holder's name: " << name << endl;
    cout << "Account balance: " << balance << endl;
  }
};

class Transaction : public Account {
private:
  string date;
  char type;
  double amount;

public:
  // Constructor
  Transaction(int num, const string& nm, double bal, const string& dt, char tp, double amt)
    : Account(num, nm, bal) {
    date = dt;
    type = tp;
    amount = amt;
  }

  // Method to read transaction details
  void read() override {
    Account::read();

    cout << "Enter transaction date: ";
    cin.ignore();
    getline(cin, date);

    cout << "Enter transaction type (C for Credit, D for Debit): ";
    cin >> type;

    cout << "Enter transaction amount: ";
    cin >> amount;
  }

  // Method to display transaction details
  void display() override {
    Account::display();
```

```
    cout << "Transaction date: " << date << endl;
    cout << "Transaction type: " << type << endl;
    cout << "Transaction amount: " << amount << endl;
  }
};

class Loan : public Account {
private:
  string date;
  double interest;

public:
  // Constructor
  Loan(int num, const string& nm, double bal, const string& dt, double intr)
    : Account(num, nm, bal) {
    date = dt;
    interest = intr;
  }

  // Method to read loan details
  void read() override {
    Account::read();

    cout << "Enter loan date: ";
    cin.ignore();
    getline(cin, date);

    cout << "Enter loan interest: ";
    cin >> interest;
  }

  // Method to display loan details
  void display() override {
    Account::display();

    cout << "Loan date: " << date << endl;
    cout << "Loan interest: " << interest << endl;
  }
};

int main() {
  Transaction transaction(123, "John Doe", 5000.0, "2023-05-20", 'C', 1000.0);
  Loan loan(456, "Alice Smith", 10000.0, "2023-05-20", 5.0);

  cout << "Transaction details:" << endl;
  transaction.display();

  cout << "\nLoan details:" << endl;
  loan.display();
```

```
  return 0;
}
```

17.(b) Write a program to overload the function add().

```
#include <iostream>
using namespace std;

// Function to add two integers
int add(int a, int b) {
  return a + b;
}

// Function to add two floating-point numbers
float add(float a, float b) {
  return a + b;
}

// Function to concatenate two strings
string add(const string& str1, const string& str2) {
  return str1 + str2;
}

int main() {
  int num1 = 5, num2 = 10;
  float float1 = 1.5, float2 = 2.5;
  string str1 = "Hello", str2 = "World";

  // Add two integers
  int sumInt = add(num1, num2);
  cout << "Sum of integers: " << sumInt << endl;

  // Add two floating-point numbers
  float sumFloat = add(float1, float2);
  cout << "Sum of floating-point numbers: " << sumFloat << endl;

  // Concatenate two strings
  string concatStr = add(str1, str2);
  cout << "Concatenated string: " << concatStr << endl;

  return 0;
}
```

18.(a) Create a Binary file and display the file contents.

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

```
using namespace std;

struct Person {
  int id;
  string name;
  int age;
};

int main() {
  // Create an array of Person objects
  Person people[] = {
    {1, "John", 25},
    {2, "Alice", 30},
    {3, "Bob", 35}
  };

  // Open the binary file for writing
  ofstream outputFile("people.dat", ios::binary);
  if (!outputFile) {
    cerr << "Error opening file for writing." << endl;
    return 1;
  }

  // Write the array of Person objects to the file
  outputFile.write(reinterpret_cast<const char*>(people), sizeof(people));
  outputFile.close();

  // Open the binary file for reading
  ifstream inputFile("people.dat", ios::binary);
  if (!inputFile) {
    cerr << "Error opening file for reading." << endl;
    return 1;
  }

  // Read and display the contents of the file
  Person person;
  while (inputFile.read(reinterpret_cast<char*>(&person), sizeof(person))) {
    cout << "ID: " << person.id << endl;
    cout << "Name: " << person.name << endl;
    cout << "Age: " << person.age << endl;
    cout << endl;
  }

  inputFile.close();

  return 0;
}
```

18.(b) Write a program to create an exception for invalid gender and catch the exception.

```
#include <iostream>
#include <exception>
using namespace std;p

class InvalidGenderException : public exception {
public:
  const char* what() const throw() {
    return "Invalid gender exception: Gender must be 'M' or 'F'.";
  }
};

void validateGender(char gender) {
  if (gender != 'M' && gender != 'F') {
    throw InvalidGenderException();
  }
}

int main() {
  char gender;

  cout << "Enter gender (M/F): ";
  cin >> gender;

  try {
    validateGender(gender);
    cout << "Gender is valid." << endl;
  }
  catch (const InvalidGenderException& ex) {
    cout << "Exception caught: " << ex.what() << endl;
  }

  return 0;
}
```