**C++ Assignment Questions**

(To be written on paper and submitted on or before 04/11/2023)

**C Programs:**

1. WAP (Write a program) to Check if a number is prime:

- Develop a program that checks if a given number is a prime number, which means it's only divisible by 1 and itself.

#include <stdio.h>

#include <math.h>

int isPrime(int a){

int j;

for (j = ceil( sqrt(a) ) ; j > 1; j--) {

if (a !== j && a % j == 0) break;

}

return j==1;

}

void main(){

int i;

clrscr();

printf("Enter a Number:");

scanf("%d",&i);

printf("Prime status of num: %d\n",isPrime(i));

getch();

}

2. WAP to sort an array of elements in ascending & descending order:

- Write a program that takes a list of numbers and sorts them in two ways: ascending order (from smallest to largest) and descending order (from largest to smallest).

#include <stdio.h>

void sort(int arr[],int flag){

int i,j,temp;

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

for(j=0;j<9-i;j++){

if(flag){

if(arr[j]>arr[j+1]){

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

else{

if(arr[j]<arr[j+1]){

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

}

}

void printArray(int arr[]){

int i;

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

printf("%d ",arr[i]);

}

printf("\n");

}

void main(){

int i;

int arr[10];

clrscr();

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

printf("Enter a Number:");

scanf("%d",&arr[i]);

}

sort(arr,1)//sort in ascending

printf("Array in ascending:\n");

printArray(arr);

sort(arr,0);//sort in descending

printf("Array in descending:\n");

printArray(arr);

getch();

}

3. WAP to find the maximum and minimum number from an array of numbers:

- Create a program that finds the largest and smallest numbers within a set of numbers provided as input.

#include <stdio.h>

int maxMin(int arr[],int flag){

int i,m;

m=arr[0];

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

if(flag){

if(arr[i]>m)

m=arr[i];

}

else

if(arr[i]<m)

m=arr[i];

}

return m;

}

void main(){

int i;

int arr[10];

clrscr();

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

printf("Enter a Number:");

scanf("%d",&arr[i]);

}

printf("Maximum is :%d.\n",maxMin(arr,1));

printf("Minimum is :%d.",maxMin(arr,0));

getch();

}

4. WAP to check if a number is palindrome:

- Design a program to determine whether a given number remains the same when its digits are reversed, making it a palindrome.

#include<stdio.h>

int isPalindrome(int i){

int rev=0;

int temp=i;

while(temp>0){

rev=rev\*10+temp%10;

temp/=10;

}

return rev==i;

}

void main()

{

int i;

clrscr();

printf("Enter a number:");

scanf("%d",&i);

printf("Number Palindrome status is: %d",isPalindrome(i));

getch();

}

5. WAP to calculate and print the factorial of a given number:

#include<stdio.h>

int factorial(int a){

if(a<0)

printf("Negative Number!!!\n");

return -1;

else if(a==0 || a==1)

return 1;

else

return a\*factorial(a-1);

}

void main()

{

int i;

clrscr();

printf("Enter a number:");

scanf("%d",&i);

printf("Factorial of %d is: %d",i,factorial(i));

getch();

}

6. WAP to display the Fibonacci series up to a given number:

- Develop a program that generates and displays the Fibonacci series up to a specified number, where series starts with two ones and each next number is the sum of the two preceding numbers.

#include<stdio.h>

void fibonacci(int a){

int first=0,second=1,temp,i;

if(a==0)

return ;

else{

for(i=1;i<=a;i++){

if(i==1)

printf("0 ");

else if(i==2)

printf("1 ");

else{

temp=first;

first=second;

second+=temp;

printf("%d ",second);

}

}

}

}

void main()

{

int i;

clrscr();

printf("Enter a number:");

scanf("%d",&i);

fibonacci(i);

getch();

}

7. WAP to demonstrate the use of different types of functions in C:

- Present a program showcasing the concept of functions in C, highlighting how to define and use them for better code organization and reusability.

#include<stdio.h>

void fun1(){

printf("fun1: NO parameter no return value.\n");

}

int fun2(){

printf("fun2: NO parameter with return value 2.\n");

return 2;

}

void fun3(int a){

printf("fun3: parameter %d with No return value.\n",a);

}

int fun4(int a){

printf("fun4: parameter %d with return value 4.\n",a);

return 4;

}

void main()

{

int a,temp;

clrscr();

printf("Enter a:");

scanf("%d",&a);

fun1();

temp=fun2();

printf("fun2 returned value to main is: %d.\n",temp);

fun3(a);

temp=fun4(a);

printf("fun4 returned value to main is: %d.\n",temp);

getch();

}

8. WAP to explain concept of swapping two variables with its outcome using three different functions that supports parameters passing by value, parameters passing by pointer, and parameter passing by reference.

#include<stdio.h>

void swap(int \*a,int \*b){

\*a = \*a+\*b;

\*b = \*a-\*b;

\*a = \*a-\*b;

}

void main()

{

int a,b;

clrscr();

printf("Enter a:");

scanf("%d",&a);

printf("Enter b:");

scanf("%d",&b);

printf("Before swapping: a:%d b:%d\n",a,b);

swap(&a,&b);

printf("After swapping: a:%d b:%d\n",a,b);

getch();

}

9. WAP to demonstrate the use of pointer to structure.

#include<stdio.h>

void main()

{

int a,b,t;

clrscr();

printf("Enter a:");

scanf("%d",&a);

printf("Enter b:");

scanf("%d",&b);

printf("Before swapping: a:%d b:%d\n",a,b);

t=a;

a=b;

b=t;

printf("After swapping: a:%d b:%d\n",a,b);

getch();

}

**C++ Programs:**

1.WAP to demonstrate the concept of class and object:

- Provide a program that demonstrates the creation of a class with its attributes and

methods, and the instantiation of objects from that class.

#include <iostream.h>

#include <conio.h>

class MyClass {

public:

int myVar;

void display() {

cout << "Value of myVar: " << myVar << endl;

}

};

void main() {

clrscr();

MyClass obj;

obj.myVar = 42;

obj.display();

getch();

}

2. WAP to demonstrate the concept of default arguments in a function:

- Illustrate the use of default arguments in C++ functions, allowing parameters to take default values if not explicitly provided.

#include <iostream.h>

#include <conio.h>

int add(int a, int b = 0) {

return a + b;

}

void main() {

clrscr();

int sum1 = add(5);

int sum2 = add(5, 3);

cout << "Sum1: " << sum1 << endl;

cout << "Sum2: " << sum2 << endl;

getch();

}

3. WAP to demonstrate the concept of single inheritance:

- Develop a program that showcases single inheritance in C++, where one class inherits attributes and methods from another class.

#include <iostream.h>

#include <conio.h>

class Base {

public:

int baseVar;

};

class Derived : public Base {

public:

int derivedVar;

};

void main() {

clrscr();

Derived obj;

obj.baseVar = 10;

obj.derivedVar = 20;

cout << "BaseVar: " << obj.baseVar << endl;

cout << "DerivedVar: " << obj.derivedVar << endl;

getch();

}

4. WAP to demonstrate the concept of multiple inheritance:

- Create an example that exhibits multiple inheritance, a C++ feature where a class can inherit from multiple parent classes.

#include <iostream.h>

#include <conio.h>

class ClassA {

public:

int varA;

};

class ClassB {

public:

int varB;

};

class MultipleInheritance : public ClassA, public ClassB {

public:

int varC;

};

void main() {

clrscr();

MultipleInheritance obj;

obj.varA = 10;

obj.varB = 20;

obj.varC = 30;

cout << "VarA: " << obj.varA << endl;

cout << "VarB: " << obj.varB << endl;

cout << "VarC: " << obj.varC << endl;

getch();

}

5. WAP to demonstrate the concept of multilevel inheritance:

- Design a program to illustrate multilevel inheritance in C++, where a class derives from a class that is derived from another class.

#include <iostream.h>

#include <conio.h>

class Grandparent {

public:

int grandparentVar;

};

class Parent : public Grandparent {

public:

int parentVar;

};

class Child : public Parent {

public:

int childVar;

};

void main() {

clrscr();

Child obj;

obj.grandparentVar = 10;

obj.parentVar = 20;

obj.childVar = 30;

cout << "GrandparentVar: " << obj.grandparentVar << endl;

cout << "ParentVar: " << obj.parentVar << endl;

cout << "ChildVar: " << obj.childVar << endl;

getch();

}

6. WAP to demonstrate the concept of hierarchical inheritance:

- Present a C++ program that demonstrates hierarchical inheritance, where multiple child classes inherit from a single parent class.

#include <iostream.h>

#include <conio.h>

class Parent {

public:

int parentVar;

};

class Child1 : public Parent {

public:

int child1Var;

};

class Child2 : public Parent {

public:

int child2Var;

};

void main() {

clrscr();

Child1 obj1;

Child2 obj2;

obj1.parentVar = 10;

obj1.child1Var = 20;

obj2.parentVar = 30;

obj2.child2Var = 40;

cout << "Child1 - ParentVar: " << obj1.parentVar << ", Child1Var: " << obj1.child1Var << endl;

cout << "Child2 - ParentVar: " << obj2.parentVar << ", Child2Var: " << obj2.child2Var << endl;

getch();

}

7. WAP to demonstrate the concept of hybrid inheritance:

- Showcase hybrid inheritance in C++, which combines multiple forms of inheritance (e.g., single, multiple, and hierarchical) in a single program.

#include <iostream.h>

#include <conio.h>

class A {

public:

void displayA() {

cout << "Class A" << endl;

}

};

class B : public A {

public:

void displayB() {

cout << "Class B" << endl;

}

};

class C {

public:

void displayC() {

cout << "Class C" << endl;

}

};

class D : public B, public C {

public:

void displayD() {

cout << "Class D" << endl;

}

};

void main() {

clrscr();

D obj;

obj.displayA();

obj.displayB();

obj.displayC();

obj.displayD();

getch();

}

8. WAP to demonstrate the concept of function overloading:

- Create a program that shows how to define multiple functions with the same name but different parameter lists, enabling polymorphism in C++.

#include <iostream.h>

#include <conio.h>

class Calculator {

public:

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

};

void main() {

clrscr();

Calculator calc;

int result1 = calc.add(5, 3);

double result2 = calc.add(2.5, 3.7);

cout << "Result 1: " << result1 << endl;

cout << "Result 2: " << result2 << endl;

getch();

}

9. WAP to demonstrate the concept of friend function and friend class:

- Explain the concept of friend functions in C++ and how they allow external functions to access private or protected members of a class.

#include <iostream.h>

#include <conio.h>

class MyFriend;

class MyClass {

private:

int x;

public:

MyClass() : x(5) {}

friend void showPrivate(MyClass obj);

friend class MyFriend;

};

class MyFriend {

public:

void showPrivate(MyClass obj) {

cout << "Private member x: " << obj.x << endl;

}

};

void showPrivate(MyClass obj) {

cout << "Private member x: " << obj.x << endl;

}

void main() {

clrscr();

MyClass obj;

MyFriend friendObj;

showPrivate(obj);

friendObj.showPrivate(obj);

getch();

}

10. WAP to demonstrate the concept of function overriding:

- Develop a program that demonstrates the overriding of a base class function in a derived class in C++.

#include <iostream.h>

#include <conio.h>

class Base {

public:

void show() {

cout << "This is the base class." << endl;

}

};

class Derived : public Base {

public:

void show() {

cout << "This is the derived class." << endl;

}

};

void main() {

clrscr();

Base b;

Derived d;

b.show(); // Calls base class function

d.show(); // Calls derived class function

getch();

}

11. WAP to demonstrate unary operator overloading by member function:

- Write a program to overload unary operators using member functions in C++ classes, enabling custom behaviour for operators like `+` or `-`.

#include <iostream.h>

#include <conio.h>

class MyNumber {

private:

int num;

public:

MyNumber() : num(0) {}

MyNumber(int n) : num(n) {}

MyNumber operator-() {

return MyNumber(-num);

}

void display() {

cout << "Number: " << num << endl;

}

};

void main() {

clrscr();

MyNumber num1(5);

MyNumber num2 = -num1;

num1.display();

num2.display();

getch();

}

12. WAP to demonstrate unary operator overloading by friend function:

- Illustrate how to overload unary operators using friend functions, which allow non-member functions to be associated with a class in C++.

#include <conio.h>

#include <iostream.h>

class MyNumber {

private:

int num;

public:

MyNumber() : num(0) {}

MyNumber(int n) : num(n) {}

friend MyNumber operator-(const MyNumber& n);

void display() {

cout << "Number: " << num << endl;

}

};

MyNumber operator-(const MyNumber& n) {

return MyNumber(-n.num);

}

void main() {

clrscr();

MyNumber num1(5);

MyNumber num2 = -num1;

num1.display();

num2.display();

getch();

}

13. WAP to demonstrate binary operator overloading by member function:

- Show how to overload binary operators using member functions in C++ classes, facilitating custom operations for operators like `+`, `-`, or `\*`.

#include <conio.h>

#include <iostream.h>

class Complex {

private:

int real;

int imag;

public:

Complex() : real(0), imag(0) {}

Complex(int r, int i) : real(r), imag(i) {}

Complex operator+(const Complex& other) {

Complex result;

result.real = real + other.real;

result.imag = imag + other.imag;

return result;

}

void display() {

cout << real << " + " << imag << "i" << endl;

}

};

void main() {

clrscr();

Complex num1(3, 2);

Complex num2(1, 4);

Complex sum = num1 + num2;

num1.display();

num2.display();

cout << "Sum: ";

sum.display();

getch();

}

14. WAP to demonstrate binary operator overloading by friend function:

- Demonstrate the overloading of binary operators using friend functions in C++, allowing custom-defined behaviour for operations like addition, subtraction, or multiplication.

#include <conio.h>

#include <iostream.h>

class Complex {

private:

int real;

int imag;

public:

Complex() : real(0), imag(0) {}

Complex(int r, int i) : real(r), imag(i) {}

friend Complex operator+(const Complex& a, const Complex& b);

void display() {

cout << real << " + " << imag << "i" << endl;

}

};

Complex operator+(const Complex& a, const Complex& b) {

Complex result;

result.real = a.real + b.real;

result.imag = a.imag + b.imag;

return result;

}

void main() {

clrscr();

Complex num1(3, 2);

Complex num2(1, 4);

Complex sum = num1 + num2;

num1.display();

num2.display();

cout << "Sum: ";

sum.display();

getch();

}

15. WAP to demonstrate the different types of constructors and destructor.

- Present a program that showcases constructors (for object initialization) and destructors (for resource cleanup) in C++ classes.

#include <iostream.h>

#include <conio.h>

class MyClass {

public:

int data;

MyClass() {

cout << "Default constructor called." << endl;

data = 0;

}

MyClass(int val) {

cout << "Parameterized constructor called." << endl;

data = val;

}

~MyClass() {

cout << "Destructor called." << endl;

}

};

void main() {

clrscr();

MyClass obj1; // Default constructor

MyClass obj2(42); // Parameterized constructor

cout << "Data in obj1: " << obj1.data << endl;

cout << "Data in obj2: " << obj2.data << endl;

getch();

}

16. WAP to demonstrate inline function:

- Develop a program that demonstrates the use of inline functions in C++, which provide faster execution by replacing function calls with the actual code.

#include <iostream.h>

#include <conio.h>

inline int square(int x) {

return x \* x;

}

void main() {

clrscr();

int num = 5;

int result = square(num);

cout << "Square of " << num << " is " << result << endl;

getch();

}

17. WAP to demonstrate virtual function:

- Create a program to illustrate the concept of virtual functions in C++, allowing dynamic binding and enabling polymorphism.

#include <conio.h>

#include <iostream.h>

class Shape {

public:

virtual void display() {

cout << "This is a shape." << endl;

}

};

class Circle : public Shape {

public:

void display() {

cout << "This is a circle." << endl;

}

};

void main() {

clrscr();

Shape\* shape = new Circle;

shape->display(); // Calls the derived class function

delete shape;

getch();

}

18. WAP to demonstrate abstract class:

- Show how to define an abstract class in C++, a class that cannot be instantiated on its own but serves as a base for derived classes.

#include <conio.h>

#include <iostream.h>

class Shape {

public:

virtual void display() = 0;

};

class Circle : public Shape {

public:

void display() {

cout << "This is a circle." << endl;

}

};

void main() {

clrscr();

// Shape shape; // You cannot instantiate an abstract class

Circle circle;

circle.display();

getch();

}

19. WAP to demonstrate different types of type casting operations:

- Provide examples of type casting in C++, including implicit and explicit type conversions between data types.

#include <conio.h>

#include <iostream.h>

void main() {

clrscr();

int num1 = 10;

double num2 = num1; // Implicit casting from int to double

cout << "Implicit Type Conversion: " << num2 << endl;

// Explicit Type Conversion

double num3 = 15.5;

int num4 = int(num3); // Explicit casting from double to int

cout << "Explicit Type Conversion: " << num4 << endl;

getch();

}

20. WAP to demonstrate static data members and member functions:

- Explain how to use static members and functions within a C++ class, which are shared among all class instances and don't require object creation.

#include <conio.h>

#include <iostream.h>

class MyClass {

public:

static int count;

int data;

MyClass() {

count++;

}

static void displayCount() {

cout << "Count: " << count << endl;

}

};

int MyClass::count = 0;

void main() {

clrscr();

MyClass obj1, obj2, obj3;

obj1.displayCount(); // Accessing static function through an object

MyClass::displayCount(); // Accessing static function through class

getch();

}

21. WAP to demonstrate templates:

- Present a program that demonstrates the use of templates in C++, enabling the creation of generic functions and classes that work with various data types.

#include <conio.h>

#include <iostream.h>

template <class T>

T add(T a, T b) {

return a + b;

}

void main() {

clrscr();

int num1 = 5, num2 = 3;

double d1 = 2.5, d2 = 1.5;

int sum1 = add(num1, num2);

double sum2 = add(d1, d2);

cout << "Sum 1: " << sum1 << endl;

cout << "Sum 2: " << sum2 << endl;

getch();

}

22. WAP to demonstrate reading data from file and writing data to file:

- Develop a program that shows how to read data from and write data to files in C++, allowing data persistence and retrieval.

#include <iostream.h>

#include <fstream.h>

int main() {

ofstream outfile("data.txt"); // Create a file for writing

if (!outfile) {

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

return 1;

}

// Writing data to the file

outfile << "Hello, World!" << endl;

outfile << "This is a Turbo C++ file handling example." << endl;

outfile.close();

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

ifstream infile("data.txt"); // Create a file for reading

if (!infile) {

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

return 1;

}

// Reading data from the file

char ch;

while (!infile.eof()) {

infile.get(ch);

cout << ch;

}

infile.close();

return 0;

}