|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

WEEK 5

**Program 5.1:**

**Create a C++ program that demonstrates the use of constructors and destructors in a class.**

**Aim:**

To develop a C++ program that demonstrates the use of constructors and destructors in a class.

# Description:

In C++, **constructors** and **destructors** are special member functions of a class:

* A **constructor** is automatically called when an object is created. It is used to initialize data members.
* A **destructor** is automatically called when an object goes out of scope or is explicitly deleted. It is used to free resources or perform cleanup tasks.

# Syntax:

class ClassName

{

private:

data\_type member1; data\_type member2;

public:

ClassName(parameters)

{

}

~ClassName()

{

}

void memberFunction()

{

}

};

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

int main()

{

ClassName object(arguments);

}

# Program

#include <iostream> using namespace std; class Student { private:

string name; int age;

public:

Student(string n, int a) { name = n;

age = a;

cout << "Constructor called for " << name << endl;

}

~Student() {

cout << "Destructor called for " << name << endl;

}

void display() {

cout << "Name: " << name << ", Age: " << age << endl;

}

};

int main() { string n; int a;

cout<<”Roll No:24B11AI439”<<endl;

cout << "Enter student name: ";

getline(cin, n);

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

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

Student s1(n, a); s1.display();

cout << "End of program.\n"; return 0;

}

# Output:

Roll No:24B11AI439

Enter student name: Anil Enter student age: 19 Constructor called for Anil Name: Sneha, Age: 19 End of program.

Destructor called for Anil.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

# Program 5.2:

**Develop a C++ program that illustrates constructor overloading. Aim:**

To develop a C++ program that illustrates constructor overloading.

# Description:

In C++, **constructor overloading** means defining multiple constructors within the same class but with different parameter lists. The compiler decides which constructor to invoke based on the number and types of arguments passed when creating an object.

In this program:

* Rectangle() → Default constructor initializes length and width to 0.
* Rectangle(int side) → Parameterized constructor for creating a **square**, assigning the same value to length and width.
* Rectangle(int l, int w) → Parameterized constructor for creating a **rectangle**, initializing length and width separately.

This demonstrates how different constructors are automatically invoked based on user choice

# Syntax:

class ClassName

{

private:

data\_type member1; data\_type member2;

public:

ClassName()

{

}

ClassName(data\_type param1)

{

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

}

ClassName(data\_type param1, data\_type param2)

{

}

};

int main()

{

ClassName obj1; ClassName obj2(value);

calls two-parameter constructor

}

# Program:

#include <iostream>

using namespace std; class Rectangle { private:

int length; int width; public:

Rectangle() { length = 0;

width = 0;

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

}

Rectangle(int side) {

length = side; width = side;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

cout << "Square constructor called!" << endl;

}

Rectangle(int l, int w) { length = l;

width = w;

cout << "Rectangle constructor called!" << endl;

}

void display() {

cout << "Length: " << length << ", Width: " << width << endl;

}

};

int main() { int choice;

cout<<”Roll No:24B11AI439”<<endl;

cout << "Choose how to create the rectangle:\n"; cout << "1. Default (0x0)\n";

cout << "2. Square (one side)\n";

cout << "3. Rectangle (length & width)\n";

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

if (choice == 1) { Rectangle r1; r1.display();

}

else if (choice == 2) {

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

int side;

cout << "Enter side length: "; cin >> side;

Rectangle r2(side); r2.display();

}

else if (choice == 3) { int l, w;

cout << "Enter length: "; cin >> l;

cout << "Enter width: "; cin >> w;

Rectangle r3(l, w); r3.display();

}

else {

cout << "Invalid choice!\n";

}

return 0;

}

# Output 1 :

Roll No:24B11AI439

Choose how to create the rectangle:

1. Default (0x0)
2. Square (one side)
3. Rectangle (length & width) Enter choice: 1

Default constructor called!

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Length: 0, Width: 0

# Output 2 :

Roll No:24B11AI439

Choose how to create the rectangle:

1. Default (0x0)
2. Square (one side)
3. Rectangle (length & width) Enter choice: 2

Enter side length: 50 Square constructor called! Length: 50, Width: 50

# Output 3:

Roll No:24B11AI439

Choose how to create the rectangle:

1. Default (0x0)
2. Square (one side)
3. Rectangle (length & width) Enter choice: 3

Enter length: 15

Enter width: 20

Rectangle constructor called! Length: 15, Width: 20

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

# Program.5.3:

**Write a C++ program that illustrates the use of a copy constructor Aim:**

To write a C++ program that illustrates the use of a **copy constructor** to initialize one object from another.

# Description:

In C++, a **copy constructor** is a special constructor used to create a new object as a copy of an existing object.

It takes a reference to another object of the same class as its parameter. The copy constructor is invoked:

* When an object is initialized from another object (Rectangle rect2 = rect1;).
* When an object is passed **by value** to a function.
* When an object is returned **by value** from a function.

By defining a custom copy constructor, we can control how the copying of object data happens, instead of relying on the compiler-generated default.

# Syntax:

class ClassName { private:

public:

ClassName(type1 arg1, type2 arg2, ...) {

}

ClassName(const ClassName &obj) {

}

};

**Program:** #include <iostream> using namespace std; class Rectangle {

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

private:

int length; int width;

public:

Rectangle(int l, int w) { length = l;

width = w;

}

Rectangle(const Rectangle &r) { length = r.length;

width = r.width;

cout << "Copy constructor called!" << endl;

}

void setLength(int l) { length = l;

}

void display() {

cout << "Length: " << length << ", Width: " << width << endl;

}

};

int main() { int l, w;

cout<<”Roll No:24B11AI439”<<endl;

cout << "Enter length and width: "; cin >> l >> w;

Rectangle rect1(l, w);

cout << "\nOriginal Rectangle (rect1):" << endl; rect1.display();

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Rectangle rect2 = rect1;

cout << "\nCopied Rectangle (rect2):" << endl; rect2.display();

rect1.setLength(l + 10);

cout << "\nAfter changing rect1's length:" << endl; cout << "rect1: ";

rect1.display(); cout << "rect2: "; rect2.display();

return 0;

}

# Output:

Roll No:24B11AI439

Enter length and width: 25 50 Original Rectangle (rect1):

Length: 25, Width: 50

Copy constructor called! Copied Rectangle (rect2): Length: 25, Width: 50

After changing rect1's length: rect1: Length: 35, Width: 50

rect2: Length: 25, Width: 50