Object-Oriented Programming with C++ - Lab Programs

1. Illustrate Class & Objects

Aim: To illustrate the concept of class and objects in C++.

Algorithm:

- Start the program.
- Define a class with data members and member functions.
- Create objects of the class.
- Access data members and member functions using objects.
- Display the output.
- Stop the program.

Program:

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

class Student {
public:
   string name;
   int rollNo;

   void display() {
      cout << "Name: " << name << endl;
      cout << "Roll No: " << rollNo << endl;
   }
};

int main() {
   Student s1; // Object creation
   s1.name = "John";
   s1.rollNo = 101;

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

   return 0;
}
```

2. Implement Member Function Defined Inside and Outside the Class

Aim: To implement member functions defined inside and outside a class.

Algorithm:

- Define a class with data members.
- Define one function inside the class and another outside using scope resolution operator ::.
- Create an object and call both functions.
- Display the results.

Program:

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

class Rectangle {
   int length, breadth;

public:
   void setData(int l, int b); // Declaration
   void area() { // Defined inside class
      cout << "Area = " << length * breadth << endl;
   }
};

// Definition outside the class
void Rectangle::setData(int l, int b) {
   length = l;
   breadth = b;
}

int main() {
   Rectangle r;
   r.setData(10, 5);
   r.area();
   return 0;
}
```

3. Demonstrate Function Overloading Applied to Member Functions

Aim: To demonstrate function overloading using member functions in a class.

Algorithm:

- Define a class.
- Declare multiple functions with the same name but different parameter lists.
- Call the overloaded functions with different arguments.
- Display the results.

Program:

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

class MathOperations {
public:
   void add(int a, int b) {
      cout << "Sum = " << a + b << endl;
   }
   void add(double a, double b) {
      cout << "Sum = " << a + b << endl;
   }
   void add(int a, int b, int c) {
      cout << "Sum = " << a + b + c << endl;
   }
};

int main() {
   MathOperations m;
   m.add(5, 10);
   m.add(2.5, 3.7);
   m.add(1, 2, 3);
   return 0;
}
```

4. Implement Passing Object as Function Argument and Returning Object from Function

Aim: To demonstrate passing objects as function arguments and returning objects from functions.

Algorithm:

- Define a class with data members.
- Create a function that accepts an object as argument.
- Perform some operations and return a new object.
- Display the results.

Program:

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

class Complex {
   int real, imag;

public:
   void getData(int r, int i) {
      real = r;
      imag = i;
   }

   Complex add(Complex c2) { // Pass object and return object
      Complex temp;
      temp.real = real + c2.real;
      temp.imag = imag + c2.imag;
      return temp;
   }

   void display() {
      cout << real << " + " << imag << "i" << endl;
   }
};

int main() {
   Complex c1, c2, c3;
   c1.getData(3, 4);
   c2.getData(5, 6);
   c3 = c1.add(c2); // Object passed and returned
   cout << "Resultant Complex Number: ";
   c3.display();
   return 0;
}
```

5. Demonstrate the Use of Constructor with Its Types and Destructor

Aim: To demonstrate constructors (default, parameterized, copy) and destructor in C++.

Algorithm:

1. Define a class with constructors and destructor.
2. Initialize objects using different constructors.
3. Display object data.
4. Observe automatic call of destructor when objects go out of scope.

Program:

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

class Sample {
  int x, y;

public:
  // Default constructor
  Sample() {
    x = 0;
    y = 0;
    cout << "Default Constructor called" << endl;
  }

  // Parameterized constructor
  Sample(int a, int b) {
    x = a;
    y = b;
    cout << "Parameterized Constructor called" << endl;
  }

  // Copy constructor
  Sample(const Sample &s) {
    x = s.x;
    y = s.y;
    cout << "Copy Constructor called" << endl;
  }

  void display() {
    cout << "x = " << x << ", y = " << y << endl;
  }

  // Destructor
  ~Sample() {
    cout << "Destructor called for object" << endl;
```

```
    }
};

int main() {
    Sample s1;          // Default constructor
    Sample s2(10, 20);   // Parameterized constructor
    Sample s3 = s2;      // Copy constructor

    s1.display();
    s2.display();
    s3.display();

    return 0;
}
```