

OBJECTIVE

- To understand the concept of classes and objects in C++.
- To implement constructors for automatic initialization of objects.
- To use destructors for releasing resources and observing object lifecycle.
- To develop modular and reusable code using object-oriented principles.

Tools and Libraries Used:

- Programming Language: C++
- IDE: Code::Blocks
- Libraries: include <iostream>, include <string>

BACKGROUND THEORY

Object-Oriented Programming in C++

C++ is an object-oriented programming (OOP) language that models real-world entities through **objects** and **classes**. OOP focuses on organizing code using reusable, modular components that encapsulate both data and behavior, promoting maintainability, scalability, and abstraction.

Object

An **object** is a real-world entity or instance of a class that combines **attributes (data)** and **behavior (functions)**. In C++, objects are the basic runtime elements used to manipulate data and interact with other components.

Key Characteristics:

- Objects encapsulate data (variables) and functions (methods) that operate on that data.
- They are created based on class definitions.
- Objects can communicate with each other through **message passing** (function calls).
- Each object has its **own copy** of class data members.
- Example: A Dog object may have attributes like color, age, weight, and behaviors like bark() and wagTail().

Class

A **class** is a blueprint or user-defined data type that defines the structure and behavior common to all its objects. It groups variables and functions into a single unit, promoting **data abstraction** and **encapsulation**.

Features:

- Defines the **template** for creating objects.

- Does **not allocate memory** until an object is instantiated.
- Supports three access specifiers:
 - private: Accessible only within the class.
 - protected: Accessible within the class and derived classes.
 - public: Accessible from outside the class.
- Facilitate **data hiding** and **information security**.

Syntax:

```
class ClassName {
private:
    // private data members
protected:
    // protected data members
public:
    // public data members and functions
};
```

Example:

```
#include <iostream>
using namespace std;
class Demo {
private:
    int a;
protected:
    int b;
public:
    int c;
    void setValues() {
        a = 10;
        b = 20;
        c = 30;
    }
};
```

```

void display() {
    cout << "Private a = " << a << endl;
    cout << "Protected b = " << b << endl;
    cout << "Public c = " << c << endl;
}
};

int main() {
    Demo obj;
    obj.setValues();
    obj.display();
    obj.c = 3; // Valid: 'c' is public
    return 0;
}

```

Constructors

A **constructor** is a special member function that is automatically called when an object is created. Its primary purpose is to initialize the data members of the object.

Characteristics:

- Has the **same name** as the class.
- **No return type** (not even void).
- **Automatically invoked** during object creation.
- If no constructor is provided, the compiler supplies a **default constructor**.
- Can be **overloaded** to support different initialization scenarios.

Types of Constructors:

1. Default Constructor

- Takes no arguments.
- Automatically provided by the compiler if no other constructors exist.

```

class Test {
    // No constructor defined
};

```

```
int main() {
    Test t1; // Default constructor is called
}
```

2. Parameterized Constructor

- Takes arguments to initialize object with specific values.

```
class Sample {
public:
    Sample(int x, int y) {
        // Custom initialization
    }
};
```

3. Copy Constructor

- Used to create a new object as a **copy of an existing object**.

```
class Sample {
public:
    Sample(const Sample &obj); // Copy constructor
};
```

Example of Constructor:

```
class Demo {
public:
    Demo() {
        cout << "Constructor called." << endl;
    }
};
```

Destructor

A **destructor** is a special member function that is automatically invoked when an object goes out of scope or is deleted. It is used to perform cleanup operations such as releasing memory, closing files, or freeing other resources.

Characteristics:

- Has the **same name** as the class, prefixed with a tilde ~.
- **No parameters** and **no return type**.

- Only **one destructor** is allowed per class.
- Automatically called when:
 - The object goes out of scope.
 - The object is explicitly deleted (in case of dynamic allocation).

Syntax:

```
~ClassName() {
    // Cleanup code
}
```

Example:

```
#include <iostream>
using namespace std;
class Demo {
public:
    Demo() {
        cout << "Constructor called." << endl;
    }
    ~Demo() {
        cout << "Destructor called." << endl;
    }
    void show() {
        cout << "Inside show function." << endl;
    }
};

int main() {
    Demo obj;    // Constructor called
    obj.show();  // Member function called
    return 0;    // Destructor called automatically
}
```

Q1. Define a class Car with private members brand, model, and year. Include public member functions to set and get these private members. Ensure that only member functions can access these private members

```

#include <iostream>
#include<string>
using namespace std;
class Car{
string brand, model;
int year;
public:
    void setData(){
        cout<<"Enter brand: ";
        getline(cin,brand);
        cout<<"Enter model: ";
        getline(cin,model);
        cout<<"Enter year: ";
        cin>>year;
    }
    void display(){
        cout<<"\n****CAR DETAILS****\n";
        cout<<"BRAND: "<<brand<<endl;
        cout<<"MODEL: "<<model<<endl;
        cout<<"YEAR: "<<year<<endl;

    }
};
int main()
{
    Car c;
    c.setData();
    c.display();
    return 0;
}

```

```
C:\work\OOP\lab3\bin\Debug x + v
Enter brand: BMW
Enter model: m2
Enter year: 2012

****CAR DETAILS****
BRAND: BMW
MODEL: m2
YEAR: 2012

Process returned 0 (0x0)   execution time : 36.081 s
Press any key to continue.
```

Q2. Define a class Book with private members title, author, and year. Implement both default and parameterized constructors. The default constructor should initialize the members with default values, and the parameterized constructor should set these values based on user input. Provide a method to display the details of a book.

```
#include <iostream>
#include<string>
using namespace std;
class Book{
    string title, author;
    int year;

public:
    Book(){
        title="";
        author="";
        year=0;
    }
    Book(string t, string a,int y){
        title=t;
        author=a;
        year=y;
    }
}
```

```

void display(){
    cout<<"\n****BOOK DETAILS****\n";
    cout<<"TITLE: "<<title<<endl;
    cout<<"AUTHOR: "<<author<<endl;
    cout<<"YEAR: "<<year<<endl;

}

};

int main()
{
    cout<<"DEFAULT CONSTRUCTOR";
    Book b;
    b.display();
    string t,a;
    int y;
    cout<<"Enter title: ";
    getline(cin,t);
    cout<<"Enter author: ";
    getline(cin,a);
    cout<<"Enter year: ";
    cin>>y;
    cin.ignore();
    Book c(t,a,y);
    c.display();
    return 0;
}

```



```
C:\work\OOP\lab3\bin\Debug  X + v
DEFAULT CONSTRUCTOR
****BOOK DETAILS****
TITLE:
AUTHOR:
YEAR: 0
Enter title: The Alchemist
Enter author: Paulo Coelho
Enter year: 1988

****BOOK DETAILS****
TITLE: The Alchemist
AUTHOR: Paulo Coelho
YEAR: 1988

Process returned 0 (0x0)   execution time : 67.368 s
Press any key to continue.
```

Q3. Create a class Employee with private members name and age. Implement a copy constructor to create a deep copy of an existing Employee object. Provide a method to display the details of an employee.

```
#include <iostream>

#include <string>

using namespace std;

class Employee {
private:
    string name;
    int age;
public:
    Employee(string n, int a) {
        name = n;
        age = a;
    }
    Employee(const Employee& c) {
        name = c.name;
        age = c.age;
    }
};
```

```

    }

    void display() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};

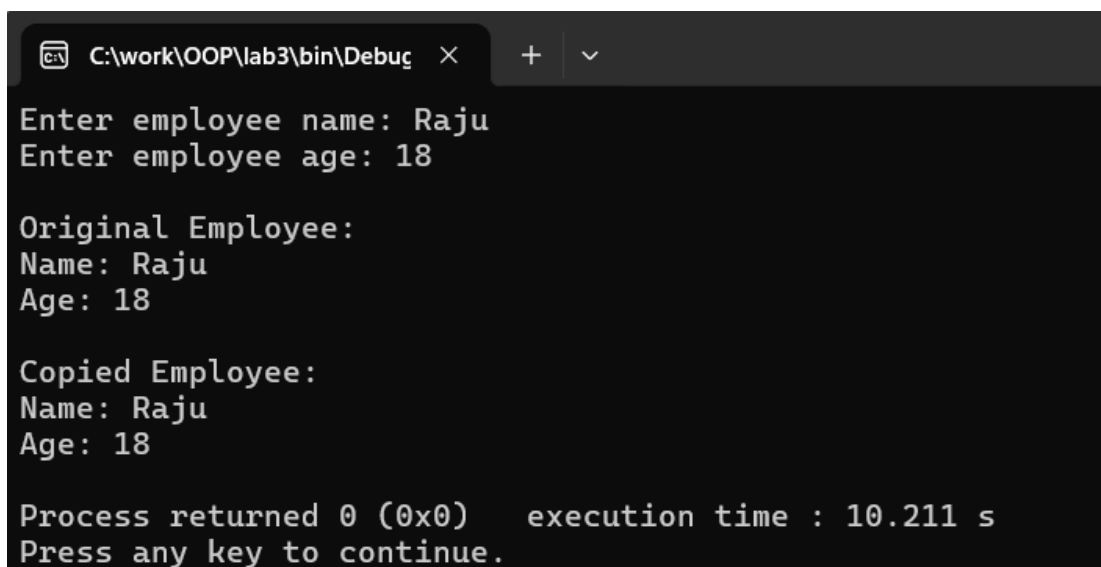
int main() {
    string name;
    int age;
    cout << "Enter employee name: ";
    getline(cin, name);
    cout << "Enter employee age: ";
    cin >> age;

    Employee emp1(name, age);
    cout << "\nOriginal Employee:\n";
    emp1.display();

    Employee emp2 = emp1;
    cout << "\nCopied Employee:\n";
    emp2.display();

    return 0;
}

```



The screenshot shows a debugger window titled "C:\work\OOP\lab3\bin\Debug" with a plus sign and a dropdown arrow. The output of the program is displayed in a black console window with white text. The output shows the user entering "Raju" for the name and "18" for the age. It then displays the details for the "Original Employee" and the "Copied Employee", both showing "Name: Raju" and "Age: 18". At the bottom, it states "Process returned 0 (0x0) execution time : 10.211 s" and "Press any key to continue."

```

C:\work\OOP\lab3\bin\Debug
Enter employee name: Raju
Enter employee age: 18

Original Employee:
Name: Raju
Age: 18

Copied Employee:
Name: Raju
Age: 18

Process returned 0 (0x0)   execution time : 10.211 s
Press any key to continue.

```

Q4. Define a class Book with a private member title. Implement a constructor that initializes title and a destructor that prints a message when the Book object is destroyed. Create instances of Book objects in main() to demonstrate the usage of the destructor.

```
#include <iostream>
#include <string>
using namespace std;
class Book {
private:
    string title;
public:
    Book(string t) {
        title = t;
        cout << "Book \"" << title << "\" is created.\n";
    }

    ~Book() {
        cout << "Book \"" << title << "\" is destroyed.\n";
    }
};
```

```

int main() {
    string t;
    cout<<"Enter title: ";
    getline(cin,t);
    Book b1(t);
    return 0;
}

```

```

C:\work\OOP\lab3\bin\Debug
Enter title: GOOD WILL
Book "GOOD WILL" is created.
Book "GOOD WILL" is destroyed.

Process returned 0 (0x0)   execution time : 14.250 s
Press any key to continue.

```

Q5. Define a class Rectangle with private members length and width. Implement a constructor to initialize these members. Write a function calculateArea() that calculates and returns the area of the rectangle. Define another function doubleDimensions(Rectangle rect) that takes a Rectangle object as an argument and doubles its length and width. Demonstrate the usage of both functions in main().

```

#include <iostream>

using namespace std;

class Rectangle {
private:
    double length;
    double width;

```

```

public:
    Rectangle(double l, double w) : length(l), width(w) {}
    double calculateArea() const {
        return length * width;
    }
    void doubleDimensions() {
        length *= 2;
        width *= 2;
    }
};

int main() {
    double l,w;
    cout<<"Enter the length and width: ";
    cin>>l>>w;
    Rectangle rect(l, w);
    cout << "Original area: " << rect.calculateArea() << endl;
    rect.doubleDimensions();
    cout << "Area after doubling dimensions: " << rect.calculateArea() << endl;
    return 0;
}

```

```

C:\work\OOP\lab3\bin\Debug
Enter the length and width: 50
12
Original area: 600
Area after doubling dimensions: 2400

Process returned 0 (0x0)   execution time : 4.786 s
Press any key to continue.

```

Q6. Define a class Student with private members name and age. Implement a constructor to initialize these members. Create an array studentArray of size 3 to store objects of

type Student. Populate the array with student details and display the details using a method displayStudentDetails().

```
#include <iostream>

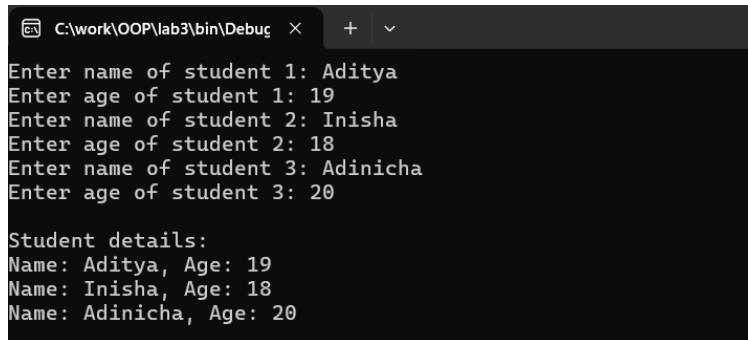
using namespace std;

class Student {
    string name;
    int age;
public:
    Student(string n, int a) : name(n), age(a) {}
    void display () {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student* s[3];

    for (int i = 0; i < 3; i++) {
        string name;
        int age;
        cout << "Enter name of student " << (i + 1) << ": ";
        getline(cin, name);
        cout << "Enter age of student " << (i + 1) << ": ";
        cin >> age;
        cin.ignore();
        s[i] = new Student(name, age);
    }
    cout << "\nStudent details:\n";
    for (int i = 0; i < 3; i++) {
        s[i]->display();
        delete s[i];
    }
}
```

```
    return 0;
}
```



```
C:\work\OOP\lab3\bin\Debug
Enter name of student 1: Aditya
Enter age of student 1: 19
Enter name of student 2: Inisha
Enter age of student 2: 18
Enter name of student 3: Adinicha
Enter age of student 3: 20

Student details:
Name: Aditya, Age: 19
Name: Inisha, Age: 18
Name: Adinicha, Age: 20
```

Q7. Define a class Math with two overloaded methods add(). The first method should take two integers as parameters and return their sum. The second method should take three integers as parameters and return their sum. Demonstrate the usage of both methods in main().

```
#include <iostream>

using namespace std;

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

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

int main() {
    Math m;
    int x, y, z;
    cout << "Enter two integers: ";
    cin >> x >> y;
    cout << "Sum of two integers: " << m.add(x, y) << endl;
    cout << "Enter three integers: ";
```

```

cin >> x >> y >> z;

cout << "Sum of three integers: " << m.add(x, y, z) << endl;

return 0;
}

```

```

C:\work\OOP\lab3\bin\Debug
Enter two integers: 20
10
Sum of two integers: 30
Enter three integers: 12
23
23
Sum of three integers: 58

Process returned 0 (0x0)   execution time : 8.716 s
Press any key to continue.

```

Q8. Implement a class Area with overloaded methods calculate(). The first method should take the radius of a circle as a parameter and return the area of the circle. The second method should take the length and width of a rectangle as parameters and return the area of the rectangle. Demonstrate the usage of both methods in main().

```

#include <iostream>

using namespace std;

class Area {
public:
    double calculate(double radius) {
        return 3.14159 * radius * radius;
    }
}

```



```

double calculate(double length, double width) {
    return length * width;
}

};

int main() {
    Area a;

    double radius = 5.0;
    double length = 10.0;
    double width = 4.0;

    double circleArea = a.calculate(radius);
    cout << "Area of Circle with radius " << radius << " is: " << circleArea << endl;

    double rectangleArea = a.calculate(length, width);
    cout << "Area of Rectangle with length " << length << " and width " << width << " is: "
    << rectangleArea << endl;

    return 0;
}

```

```

Area of circle with radius 5: 78.5
Area of rectangle with length 4 and width 6: 24

```

Conclusion:

This lab really helped us get comfortable with the basics of object-oriented programming in C++. We got hands-on experience with important concepts like classes, objects, constructors, destructors, and method overloading. Writing the actual code made it easier to understand how constructors simplify setting up objects, how destructors handle cleanup, and how access specifiers help protect data. Each program we worked on showed us the value of keeping code modular and reusable. Overall, this lab gave us a solid start with OOP and made us feel more confident about tackling more advanced programming challenges in the future.