OBJECTIVE

- To understand the working process of classes and objects.
- To initialize constructors and destructors for their own purpose.
- To enhance the concept of object and classes along with their uses.
- To promote code reusability using OOP features.

BACKGROUND THEORY

Object-Oriented Programming was invented with the purpose of solving the problems caused by previous programming language. It contains various concepts and building blocks that made the coders and developers more easier to code. Objects, classes, constructor, destructor, encapsulation, polymorphism, etc are the essential keys for OOP.

Object

An object is a real-world or logical entity that contains both data (attributes) and operations (methods). Objects are the basic runtime entities in OOP and are created using classes. Multiple objects can be created by a single class Objects are the basic runtime entities which may be created or destroyed at run time.

Class:

A class is a blueprint or template from which objects are created. It defines the data and the operations that can be performed on that data. Class has three access specifiers: public, private and protected. So, class promotes the concept of data hiding. For example, car is a class and color and drive are its members.

Syntax Example in C++:

```
class Car {
private:
string color;
public:
void driver( )
{
//driving logic
}
```

Constructors:

A constructor is a specialized function of a class that has the same name as the class and no return type. It is automatically invoked at the time of object creation to assign initial values to the data members of the class.

Function of constructor:

- To initialize object automatically when they are created.
- To avoid manual initialization using a separate

Types of constructors:

a. Default constructor
   It is a constructor that takes no arguments. It is used to initialize object with default values when no specific values are passed.

Sytanx:

```
class  ClassName{
public:
ClassName( );
};
```

b. Parameterized constructor:
   It is a constructor that takes arguments/parameters. It is used to initialize objects with specific values at the time of creation.

Syntax:

```
class ClassName {
public:
ClassName(data_type parameter1, data_type parameter2, ...);
};
```

c. Copy constructor:
   A copy constructor in C++ is a special constructor that creates a new object by initializing it with an existing object of the same class. It takes a reference to a constant object of the same class as its parameter and is used to perform member-wise copying of data.

Syntax:

```
class ClassName {
public:
ClassName(const ClassName &obj);
};
```

Destructor:

In C++, a destructor is a special member function that is automatically called when an object goes out of scope or is explicitly deleted. Its primary purpose is to free resources that the object may have acquired during its lifetime, such as memory, file handles, or network connections. Its name is as same as the class and takes no argument and return type.

Functions of destructor:

- To perform clean-up tasks (e.g., releasing memory, closing files).
- To avoid memory leaks in programs using dynamic memory.

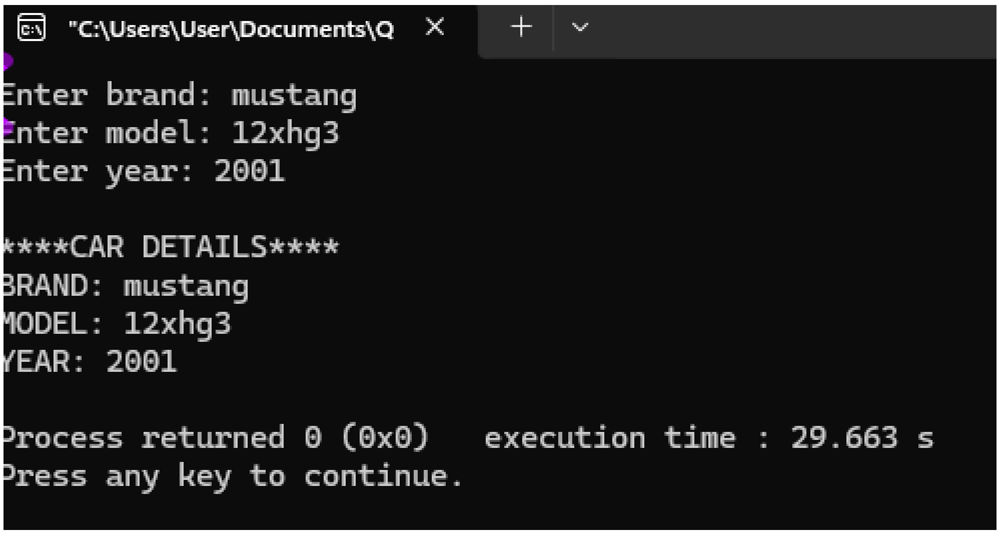
Syntax:

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

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 6

```
#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;
}
```

Output:



```
Enter brand: mustang
Enter model: 12xhg3
Enter year: 2001

****CAR DETAILS****
BRAND: mustang
MODEL: 12xhg3
YEAR: 2001

Process returned 0 (0x0)   execution time : 29.663 s
Press any key to continue.
```

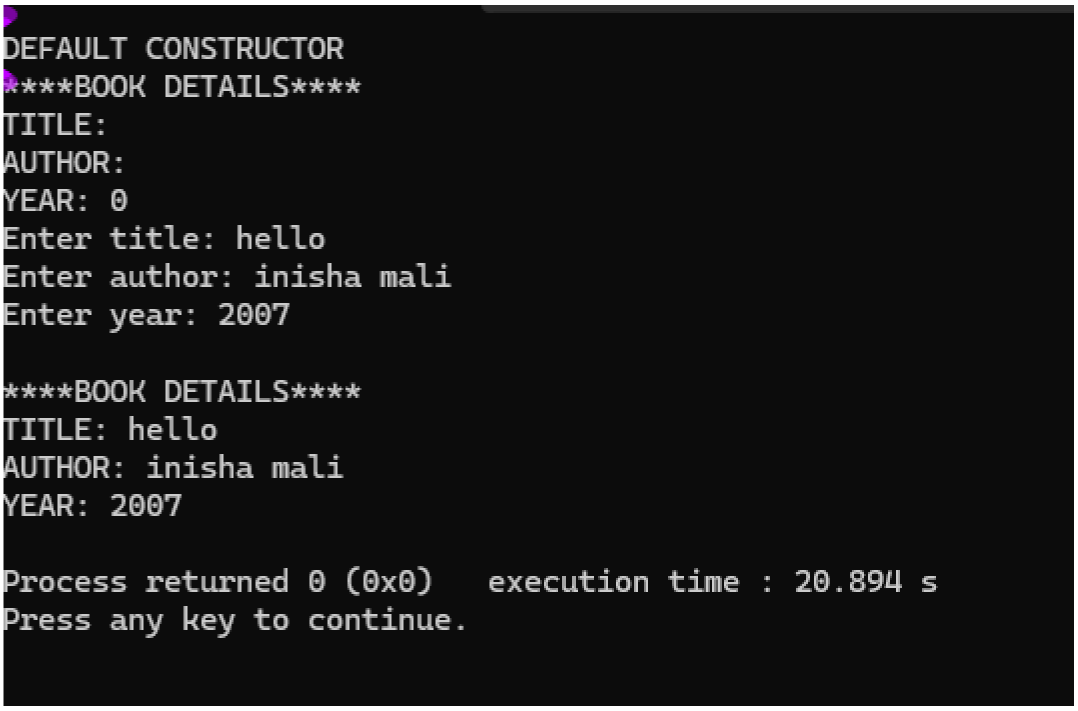
Q2. Define a class Book with private members title, author, and year. Implement both default and parameterized constructors. The default constructor should initialize the members with default values, and the parameterized constructor should set these values based on user input. Provide a method to display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

Output

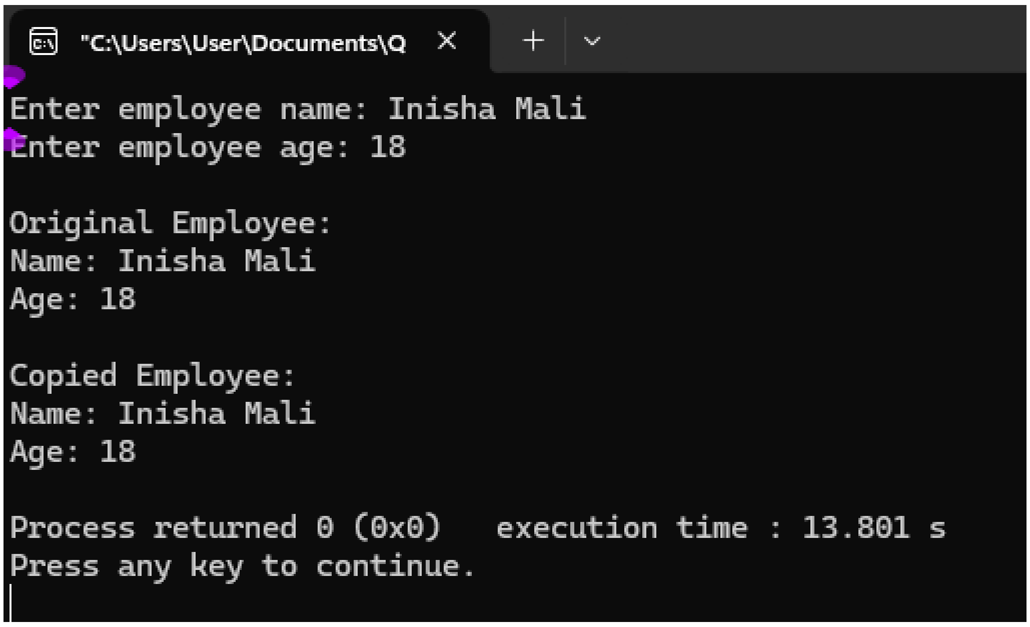


```
DEFAULT CONSTRUCTOR
****BOOK DETAILS****
TITLE:
AUTHOR:
YEAR: 0
Enter title: hello
Enter author: inisha mali
Enter year: 2007

****BOOK DETAILS****
TITLE: hello
AUTHOR: inisha mali
YEAR: 2007

Process returned 0 (0x0)   execution time : 20.894 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#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;
}
```

Output

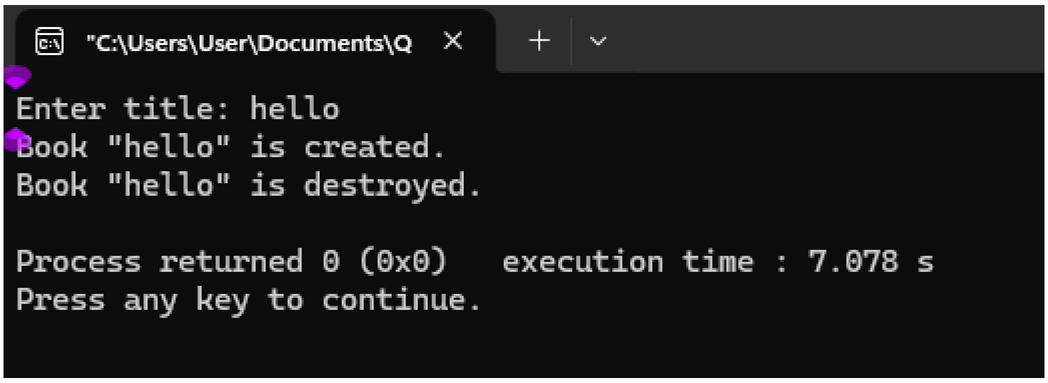


```
Enter employee name: Inisha Mali
Enter employee age: 18

Original Employee:
Name: Inisha Mali
Age: 18

Copied Employee:
Name: Inisha Mali
Age: 18

Process returned 0 (0x0)   execution time : 13.801 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;

}
```

Output



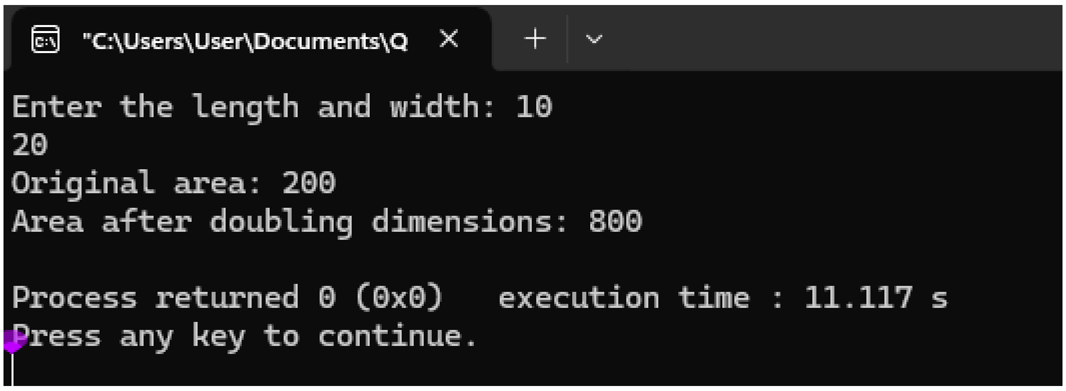
```
Enter title: hello
Book "hello" is created.
Book "hello" is destroyed.

Process returned 0 (0x0)   execution time : 7.078 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;
}
```

Output

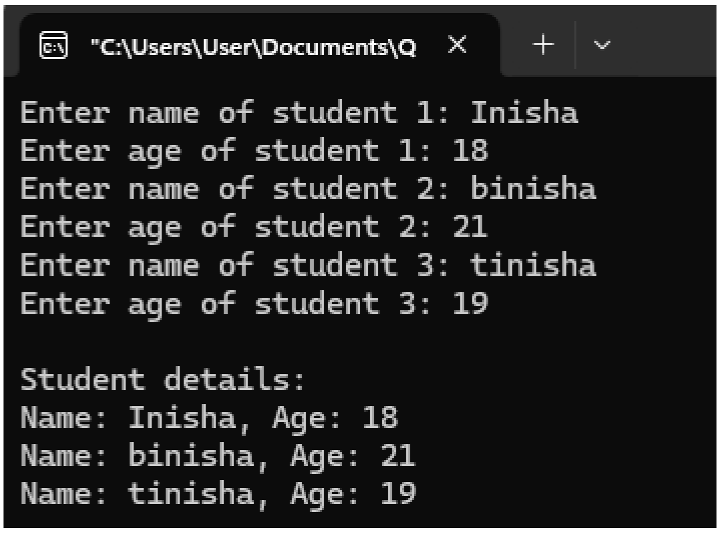


```
Enter the length and width: 10
20
Original area: 200
Area after doubling dimensions: 800

Process returned 0 (0x0)   execution time : 11.117 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;
}
```

Output



```
Enter name of student 1: Inisha
Enter age of student 1: 18
Enter name of student 2: binisha
Enter age of student 2: 21
Enter name of student 3: tinisha
Enter age of student 3: 19

Student details:
Name: Inisha, Age: 18
Name: binisha, Age: 21
Name: tinisha, Age: 19
```

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: ";
```

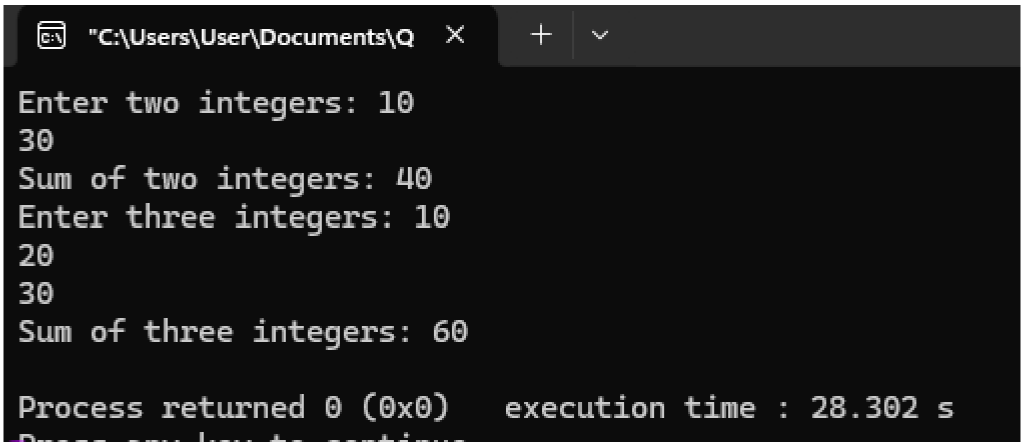
cin >> x >> y >> z;

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

return 0;

}

Output



```
"C:\Users\User\Documents\Q

Enter two integers: 10
30
Sum of two integers: 40
Enter three integers: 10
20
30
Sum of three integers: 60

Process returned 0 (0x0)   execution time : 28.302 s
```

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;

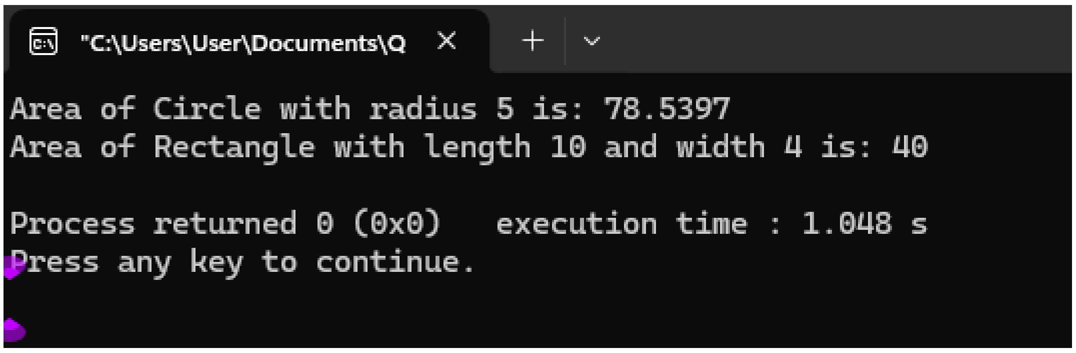
double rectangleArea = a.calculate(length, width);

cout << "Area of Rectangle with length " << length << " and width " << width << " is: " << rectangleArea << endl;

return 0;

}

Output



```
Area of Circle with radius 5 is: 78.5397
Area of Rectangle with length 10 and width 4 is: 40

Process returned 0 (0x0)   execution time : 1.048 s
Press any key to continue.
```

Conclusion:

This lab provided us with a strong introduction to the fundamentals of object-oriented programming (OOP) in C++. We gained practical experience with core concepts such as classes, objects, constructors, destructors, and method overloading. Implementing these features in code helped us better understand their real-world applications—for instance, how constructors streamline object initialization, destructors manage resource cleanup, and access specifiers ensure data security and encapsulation. Each program highlighted the importance of writing modular, reusable code..