

Department of Computer Science(BS 6th induction)

Lab Manual(04)

Roll #:24017119-003

subject
code:cs-
103- s25-E



I

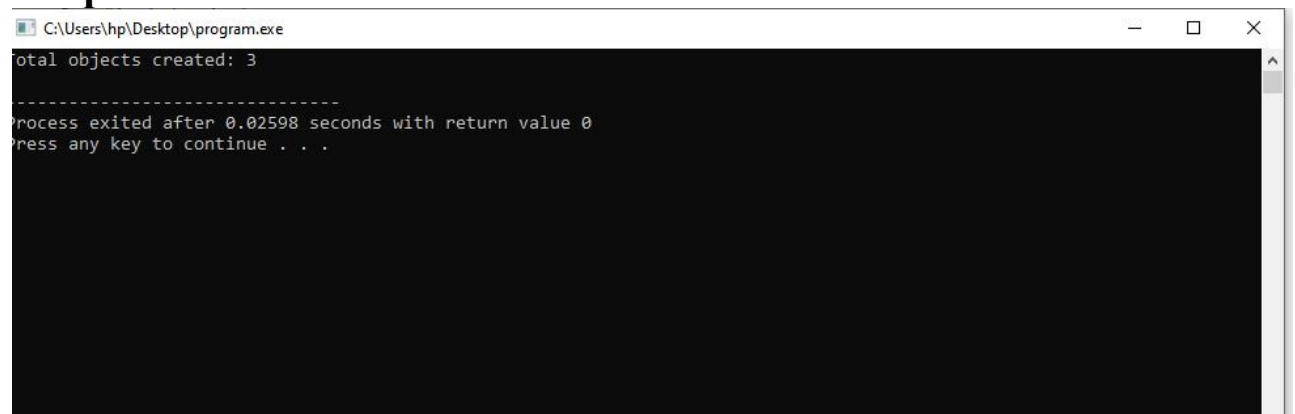
<i>Subject Name:</i>	“Object Oriented Programming”
<i>Submitted by:</i>	“Alishba Yaqoob”
<i>Submitted to:</i>	“DR.Usman Ali”
<i>Submission Date:</i>	“7,March,2025”

///

Lab Manual 04: Static Variables and Functions in C++

```
#include <iostream>
using namespace std;
class Counter {
public:
static int count; // Static member variable
Counter() {
count++; // Increment count each time an object is created
}
void display() {
cout << "Total objects created: " << count << endl;
}
};
// Initialize static member variable
int Counter::count = 0;
int main() {
Counter obj1, obj2, obj3;
obj1.display(); // Output: Total objects created: 3
return 0;
}
```

Output:



```
C:\Users\hp\Desktop\program.exe
Total objects created: 3
-----
Process exited after 0.02598 seconds with return value 0
Press any key to continue . . .
```

Lab Task:

```
#include<iostream>

using namespace std;
```

```
class Student {
private:
    int eng, urdu, math;

public:
    static int countDestroyed;
    Student(int e, int u, int m) {
        eng = e;
        urdu = u;
        math = m;
        cout << "Student created with marks: English = " << eng << ", Urdu = " << urdu << ", Math = " << math << endl;
    }
    ~Student() {
        countDestroyed++;
        cout << "Student object destroyed!" << endl;
    }
    void displayMarks() {
        cout << "English: " << eng << ", Urdu: " << urdu << ", Math: " << math << endl;
    }
    static void displayDestroyedCount() {
        cout << "Number of destroyed objects: " << countDestroyed << endl;
    }
};

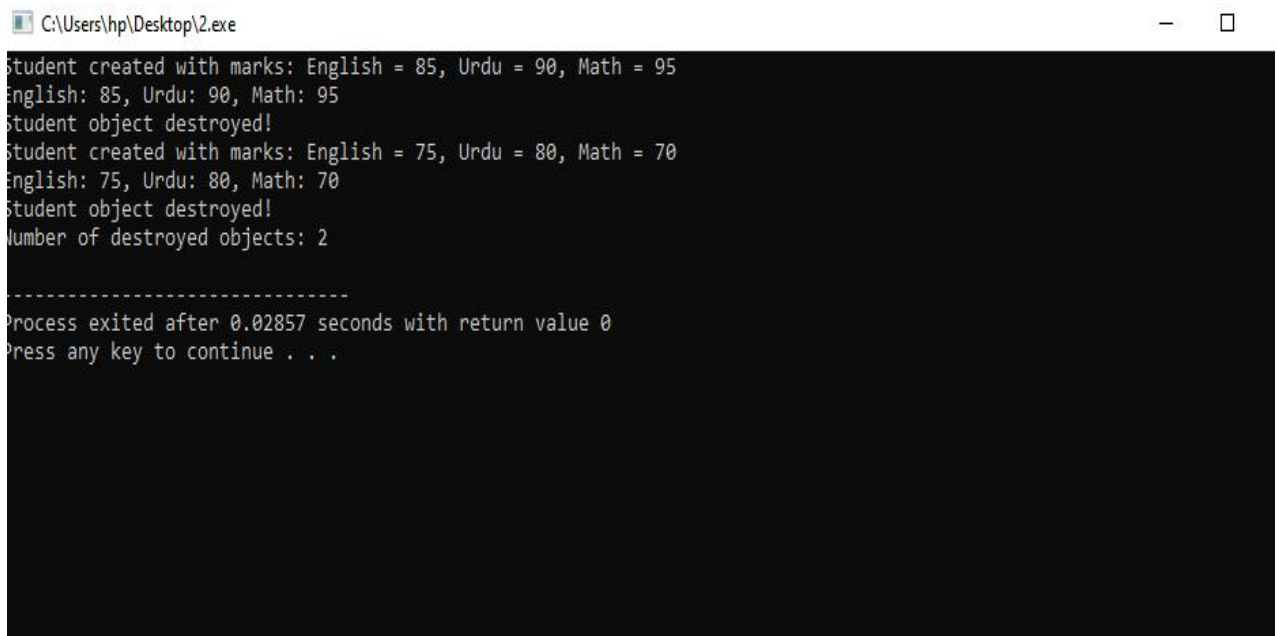
int Student::countDestroyed = 0;
```

```

int main() {
    {
        Student student1(85, 90, 95);
        student1.displayMarks();
    }
    {
        Student student2(75, 80, 70);
        student2.displayMarks();
    }
    Student::displayDestroyedCount();

    return 0;
}

```



The screenshot shows a Windows command prompt window titled "C:\Users\hp\Desktop\2.exe". The output of the program is as follows:

```

Student created with marks: English = 85, Urdu = 90, Math = 95
English: 85, Urdu: 90, Math: 95
Student object destroyed!
Student created with marks: English = 75, Urdu = 80, Math = 70
English: 75, Urdu: 80, Math: 70
Student object destroyed!
Number of destroyed objects: 2

-----
Process exited after 0.02857 seconds with return value 0
Press any key to continue . . .

```

Exercise 2: Static Local Variable

```

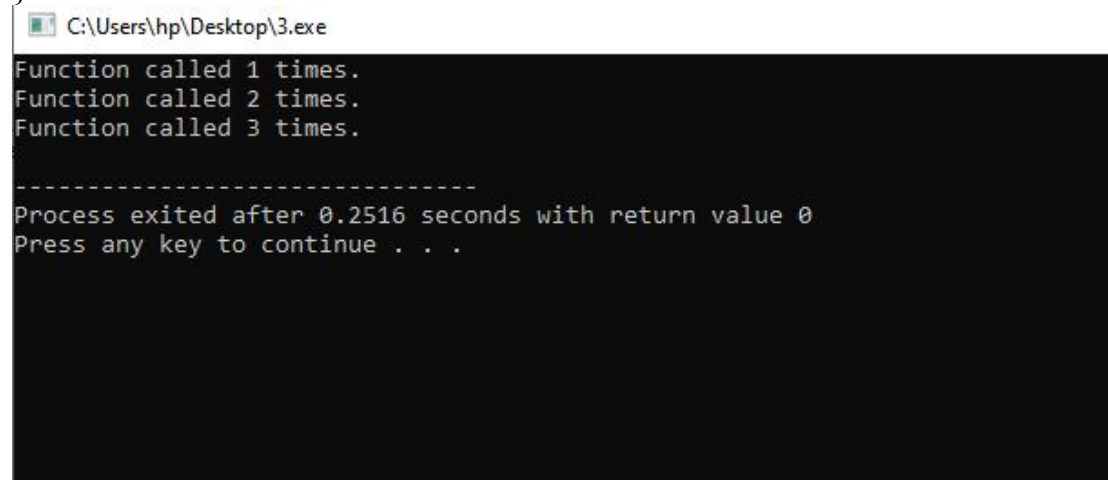
#include <iostream>
using namespace std;

```

```

void function() {
static int callCount = 0; // Static local variable
callCount++;
cout << "Function called " << callCount << " times." << endl;
}
int main() {
function(); // Output: Function called 1 times.
function(); // Output: Function called 2 times.
function(); // Output: Function called 3 times.
return 0;
}

```



```

C:\Users\hp\Desktop\3.exe
Function called 1 times.
Function called 2 times.
Function called 3 times.
-----
Process exited after 0.2516 seconds with return value 0
Press any key to continue . . .

```

Task:

Modify the function to reset the counter after 5 calls

```

#include<iostream>
using namespace std;

```

```

class Student {
private:
    int eng, urdu, math;

```

```

public:

```

```

    static int countDestroyed;
    static int destructionCount;

```

```

    Student(int e, int u, int m) {
        eng = e;
        urdu = u;
        math = m;
    }

```

```
        cout << "Student created with marks: English = " << eng << ", Urdu  
= " << urdu << ", Math = " << math << endl;  
    }
```

```
~Student() {  
    destructionCount++;  
    countDestroyed++;  
}
```

```
if (destructionCount == 5) {  
    cout << "Resetting destroyed count after 5 objects." << endl;  
    countDestroyed = 0;  
    destructionCount = 0;  
}
```

```
    cout << "Student object destroyed!" << endl;  
}
```

```
void displayMarks() {  
    cout << "English: " << eng << ", Urdu: " << urdu << ", Math: " <<  
math << endl;  
}
```

```
static void displayDestroyedCount() {  
    cout << "Number of destroyed objects: " << countDestroyed << endl;  
}  
};
```

```
int Student::countDestroyed = 0;  
int Student::destructionCount = 0;
```

```
int main() {  
  
    {  
        Student student1(85, 90, 95);  
        student1.displayMarks();  
    }  
  
    {
```

```
        Student student2(75, 80, 70);
        student2.displayMarks();
    }

    {
        Student student3(60, 65, 70);
        student3.displayMarks();
    }

    {
        Student student4(90, 80, 85);
        student4.displayMarks();
    }

    {
        Student student5(78, 82, 88);
        student5.displayMarks();
    }
    {
        Student student6(72, 74, 68);
        student6.displayMarks();
    }

    {
        Student student7(88, 92, 94);
        student7.displayMarks();
    }

    Student::displayDestroyedCount();

    return 0;
}
```

C:\Users\hp\Desktop\4.exe

```
Student created with marks: English = 85, Urdu = 90, Math = 95
English: 85, Urdu: 90, Math: 95
Student object destroyed!
Student created with marks: English = 75, Urdu = 80, Math = 70
English: 75, Urdu: 80, Math: 70
Student object destroyed!
Student created with marks: English = 60, Urdu = 65, Math = 70
English: 60, Urdu: 65, Math: 70
Student object destroyed!
Student created with marks: English = 90, Urdu = 80, Math = 85
English: 90, Urdu: 80, Math: 85
Student object destroyed!
Student created with marks: English = 78, Urdu = 82, Math = 88
English: 78, Urdu: 82, Math: 88
Resetting destroyed count after 5 objects.
Student object destroyed!
Student created with marks: English = 72, Urdu = 74, Math = 68
English: 72, Urdu: 74, Math: 68
Student object destroyed!
Student created with marks: English = 88, Urdu = 92, Math = 94
English: 88, Urdu: 92, Math: 94
Student object destroyed!
Number of destroyed objects: 2
-----
```

Exercise 3: Static Function

```
#include <iostream>
using namespace std;
class MathUtility {
public:
    static int square(int x) {
        return x * x;
    }
};
int main() {
    int num = 5;
    cout << "Square of " << num << " is " << MathUtility::square(num) <<
endl;
    return 0;
}
```


C:\Users\hp\Desktop\5.exe

Square of 5 is 25

Process exited after 0.2476 seconds with return value 0

Press any key to continue . . .

Task:

Add another static function to calculate the cube of a number.

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

```
class Student {
private:
    int eng, urdu, math;
```

```
public:
    static int countDestroyed;
    static int destructionCount;
```

```
    Student(int e, int u, int m) {
        eng = e;
        urdu = u;
        math = m;
        cout << "Student created with marks: English = " << eng
        << ", Urdu = " << urdu << ", Math = " << math << endl;
    }
```

```
    ~Student() {
        destructionCount++;
        countDestroyed++;

        if (destructionCount == 5) {
```

```

        cout << "Resetting destroyed count after 5 objects." <<
endl;
        countDestroyed = 0;
        destructionCount = 0;
    }

    cout << "Student object destroyed!" << endl;
}

void displayMarks() {
    cout << "English: " << eng << ", Urdu: " << urdu << ",
Math: " << math << endl;
}

static void displayDestroyedCount() {
    cout << "Number of destroyed objects: " <<
countDestroyed << endl;
}

static int cube(int number) {
    return number * number * number;
}
};

int Student::countDestroyed = 0;
int Student::destructionCount = 0;

int main() {
    {
        Student student1(85, 90, 95);
        student1.displayMarks();
    }

    {
        Student student2(75, 80, 70);
        student2.displayMarks();
    }
}

```

```

    {
        Student student3(60, 65, 70);
        student3.displayMarks();
    }

    {
        Student student4(90, 80, 85);
        student4.displayMarks();
    }

    {
        Student student5(78, 82, 88);
        student5.displayMarks();
    }

    {
        Student student6(72, 74, 68);
        student6.displayMarks();
    }

    {
        Student student7(88, 92, 94);
        student7.displayMarks();
    }

    Student::displayDestroyedCount();

    int num = 5;
    cout << "The cube of " << num << " is: " <<
Student::cube(num) << endl;

    return 0;
}

```

C:\Users\hp\Desktop\6.exe

```
Student created with marks: English = 85, Urdu = 90, Math = 95
English: 85, Urdu: 90, Math: 95
Student object destroyed!
Student created with marks: English = 75, Urdu = 80, Math = 70
English: 75, Urdu: 80, Math: 70
Student object destroyed!
Student created with marks: English = 60, Urdu = 65, Math = 70
English: 60, Urdu: 65, Math: 70
Student object destroyed!
Student created with marks: English = 90, Urdu = 80, Math = 85
English: 90, Urdu: 80, Math: 85
Student object destroyed!
Student created with marks: English = 78, Urdu = 82, Math = 88
English: 78, Urdu: 82, Math: 88
Resetting destroyed count after 5 objects.
Student object destroyed!
Student created with marks: English = 72, Urdu = 74, Math = 68
English: 72, Urdu: 74, Math: 68
Student object destroyed!
Student created with marks: English = 88, Urdu = 92, Math = 94
English: 88, Urdu: 92, Math: 94
Student object destroyed!
Number of destroyed objects: 2
The cube of 5 is: 125

-----
Process exited after 0.2573 seconds with return value 0
Press any key to continue . . .
```

Exercise 4: Singleton Design Pattern

```
#include <iostream>
```

```
using namespace std;
```

```
class Database {
```

```
private:
```

```
    static Database* instance; // Static pointer to the single instance
```

```
    string data;
```

```
    // Private constructor to prevent instantiation
```

```
    Database() {
```

```
        data = "Initial Data";
```

```
    }
```

```
public:
```

```
    // Static function to access the single instance
```

```
    static Database* getInstance() {
```

```
        if (!instance) {
```

```
            instance = new Database(); // Create the instance if not already
```

```
created
```

```
        }
```

```
        return instance;
```

```
    }
```

```

// Setter for data
void setData(string newData) {
    data = newData;
}

// Display data
void displayData() const {
    cout << "Data: " << data << endl;
}

// Destructor (not necessary in this simple case but included for cleanup)
~Database() {
    // Optional clean-up
}
};

// Initialize the static member outside the class
Database* Database::instance = nullptr;

int main() {
    // Access the single instance of Database
    Database* db = Database::getInstance();
    db->displayData(); // Output: Data: Initial Data
    db->setData("New Data");
    db->displayData(); // Output: Data: New Data

    // Attempt to create another instance
    Database* anotherDb = Database::getInstance();
    anotherDb->displayData(); // Output: Data: New Data (same as above)

    return 0;
}

```

Output:

```

Data: Initial Data
Data: New Data
Data: New Data

```

Task

Modify the Singleton class to include a static counter that tracks the number of times the instance

```

#include <iostream>
using namespace std;

class Singleton {
private:
    static Singleton* instance;
    static int accessCount;

    Singleton() {
        cout << "Singleton instance created." << endl;
    }

public:
    static Singleton* getInstance() {
        if (!instance) {
            instance = new Singleton();
        }
        accessCount++;
        return instance;
    }

    static int getAccessCount() {
        return accessCount;
    }

    ~Singleton() {
        cout << "Singleton instance destroyed." << endl;
    }
};

Singleton* Singleton::instance = nullptr;
int Singleton::accessCount = 0;

int main() {
    Singleton* s1 = Singleton::getInstance();
    cout << "Access count: " << Singleton::getAccessCount() << endl;

    Singleton* s2 = Singleton::getInstance();
    cout << "Access count: " << Singleton::getAccessCount() << endl;

    Singleton* s3 = Singleton::getInstance();
    cout << "Access count: " << Singleton::getAccessCount() << endl;
}

```

```
    return 0;
}
```

Output:

Singleton instance created. Access count: 1 Access count: 2
Access count: 3 Singleton instance destroyed.

Lab Report

1. Write a brief explanation of static variables and functions.
2. Include the code and output for each exercise.
3. Answer the following questions:
 - o What is the difference between a static member variable and a non-static member variable?
 - o Why can't static functions access non-static members of a class?
 - o How does the Singleton pattern ensure only one instance of a class is created?

1. Write a brief explanation of static variables and functions.

1. Explanation of Static Variables and Functions

In C++, **static variables** and **static functions** are members of a class that have a special property: they are shared across all instances of the class. Instead of each object of the class having its own copy of the variable or function, all instances of the class share the same static variable or function

Static member variable: This is a variable that is shared by all instances of the class. It is not tied to any specific instance of the class. Static variables are initialized only once and are accessible using the class name or any instance of the class.

Static member function: A static function belongs to the class itself rather than to any specific object. It can be called using the class name, and it can only access static variables or other static functions of the class. It cannot access non-static members or functions because they belong to specific instances of the class

2. Include the code and output for each exercise.

```
.#include <iostream>

using namespace std;

class Singleton {
private:
    static Singleton* instance; // Static pointer to the single instance

    static int accessCount; // Static counter to track the number of times the
instance is accessed

    Singleton() {
        cout << "Singleton instance created." << endl;
    }

public:
    static Singleton* getInstance() {
        if (!instance) {
            instance = new Singleton();
        }

        accessCount++;

        return instance;
    }

    static int getAccessCount() {
        return accessCount;
    }
}
```



```

~Singleton() {
    cout << "Singleton instance destroyed." << endl;
}

};

Singleton* Singleton::instance = nullptr;

int Singleton::accessCount = 0;

int main() {
    Singleton* s1 = Singleton::getInstance();
    cout << "Access count: " << Singleton::getAccessCount() << endl;

    Singleton* s2 = Singleton::getInstance();
    cout << "Access count: " << Singleton::getAccessCount() << endl;

    Singleton* s3 = Singleton::getInstance();
    cout << "Access count: " << Singleton::getAccessCount() << endl;

    return 0;
}

```

3. Answer the Following Questions:

1. What is the difference between a static member variable and a non-static member variable?

-

Static Member Variable:

-
- A **static member variable** is shared among all instances of a class. It has only one copy in memory, regardless of how many objects of the class are created.
- It is initialized only once, and it exists even if no instances of the class are created.
- A static member variable can be accessed using the class name or through any instance of the class.
- It is commonly used for keeping track of data or states that are common to all objects of the class (e.g., a counter).
-

Non-static Member Variable:

-
- A **non-static member variable** is unique to each instance of the class. Every object has its own copy of non-static variables.
- These variables are tied to a specific object of the class and cannot be shared across instances.
- Non-static member variables are accessed using the object of the class, and each instance can have different values for these variables.

2. Why can't static functions access non-static members of a class?

A **static function** is tied to the class itself, not to any specific instance of the class. This means:

- **Static functions do not have access to instance-specific data** (non-static members) because they do not operate on an instance of the class. They are designed to work with **class-wide data**, which includes static variables or other static methods.
- Non-static members belong to a specific instance of the class. Since static functions don't have access to any specific instance (they don't know which instance to refer to), they cannot access or modify instance-specific data (non-static members).

In simple terms: **Static functions operate at the class level**, and non-static members belong to a specific instance, so static functions don't know about or have access to them.

3. How does the Singleton pattern ensure only one instance of a class is created?

The **Singleton pattern** ensures that only one instance of a class is created by following these key steps:

1. Private Constructor:

- The class's constructor is made **private** so that objects cannot be created directly from outside the class.

2. **Static Pointer to the Instance:**

- A **static pointer** to the single instance of the class is maintained. This pointer is initially set to nullptr (indicating that the instance does not exist).

3. **Static getInstance() Method:**

- The getInstance() method is a **static method** that checks if the instance already exists:
 - If the instance doesn't exist (i.e., the static pointer is nullptr), it creates the instance and stores it in the static pointer.
 - If the instance already exists, it simply returns the existing instance.

4. **Ensures One Instance:**

- The static pointer ensures that there is only one instance of the class. Even if the getInstance() method is called multiple times, it will always return the same instance. No new instance is created after the first one.

In summary, the Singleton pattern **controls the creation of objects** and ensures that only **one instance** of the class is created by using a static pointer and a private constructor, and it only provides access to this single instance through a static method.