Lab Manual

CS164 – Object Oriented Programming Lab
Lab No: **05**
Topic: **Constant and Static Data Members, This pointer**

Class: **BSSE**
Semester: **II - A**
Session: **Spring, 2021**
Instructor: **Dr. Abdul Haleem**
Lab Instructor: **Ms. Saima Majeed**

Lab Date: **March 31st, 2021**
Lab Time: **02:40hrs – 5:30hrs**

Air University Islamabad
FACULTY OF COMPUTING & ARTIFICIAL INTELLIGENCE

Department of Creative Technologies

Instructions

Submission: Use proper naming convention for your submission file. Name the submission file as LabNO_ROLLNUM (e.g. Lab01_00000). Submit the file on Google Classroom within the deadline. Failure to submit according to the above format would result in deduction of 10% marks. Submissions on the email will not be accepted.

Plagiarism: Plagiarism cases will be dealt with strictly. If found plagiarized, both the involved parties will be awarded zero marks in the assignment, all of the remaining assignments, or even an F grade in the course. Copying from the internet is the easiest way to get caught!

Deadline: The deadlines to submit the assignment are hard. Late submission with marks deduction will be accepted according to the course policy shared by the instructor. Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.

Comments: Comment your code properly. Bonus marks (maximum 10%) will be awarded to well comment code. Write your name and roll number (as a block comment) at the beginning of the solution to each problem.

Tip: For timely completion of the assignment, start as early as possible. Furthermore, work smartly - as some of the problems can be solved using smarter logic.

1. Note: Follow the given instructions to the letter, failing to do so will result in a zero.

Objectives

In this lab, you will learn:

- Constant Data Members, Constant Member Functions, and Constant Objects
- Static Data Members and its initialization and accessing
- “This” Pointer

Concepts

1. Constant Data Members:

Data members of a class may be declared as const. Such a data member **must** be initialized by the constructor using an initialization list. Once initialized, a const data member may **never modified**, not even in the **constructor** or **destructor**. Data members that are both **static** and **const** have their own rules for initialization. Furthermore, a const data member cannot be **initialized** at the time of declaration or within the member function definition. To **initialize a const data member** of a class, follow given **syntax**:

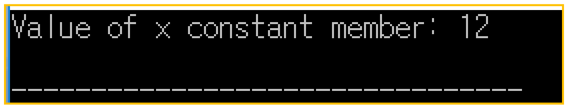
```
const data_type constant_member_name;
```

The following **example** illustrates the **initialization** and **declaration** of constant data members:

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

class ConstDataMember
{
    private:
        // While declaring constant member, you need to specify
        // 'const' keyword
        const int x;
    public:
        //const initialization
        ConstDataMember():x(12){}
        //print function
        void display()
        {
        cout<<"Value of x constant member: "<<x<<endl;
        }
};
int main()
{
    ConstDataMember obj1;
    obj1.display();

    return 0;
}
```

In this program, **Number** is a class and **x** is a constant integer data member, we are initializing it with 12.

Output:



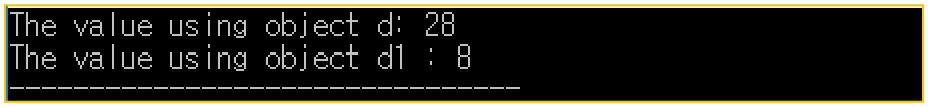
```
Value of x constant member: 12
--------------------------------
```

2. Constant Member Functions:

The const member functions are the functions which are declared as constant in the program. The object called by these functions cannot be modified. It is recommended to use const keyword so that accidental changes to object are avoided. A **const member function** can be called by **any type of object**. **Non-const functions** can be called by **non-const objects** only. The following is an **example** that illustrates the **syntax** of constant member functions:

```cpp
#include<iostream>
using namespace std;
class Demo {
   int val;
   public:
   Demo(int x = 0) {
      val = x;
   }
   // The const member function must use 'const' keyword
   int getValue() const {
      return val;
   }
};
int main() {
   const Demo d(28);
   Demo d1(8);
   // We can access the const member function with any type of object
   cout << "The value using object d: " << d.getValue();
   cout << "\nThe value using object d1 : " << d1.getValue();
   return 0;
}
```

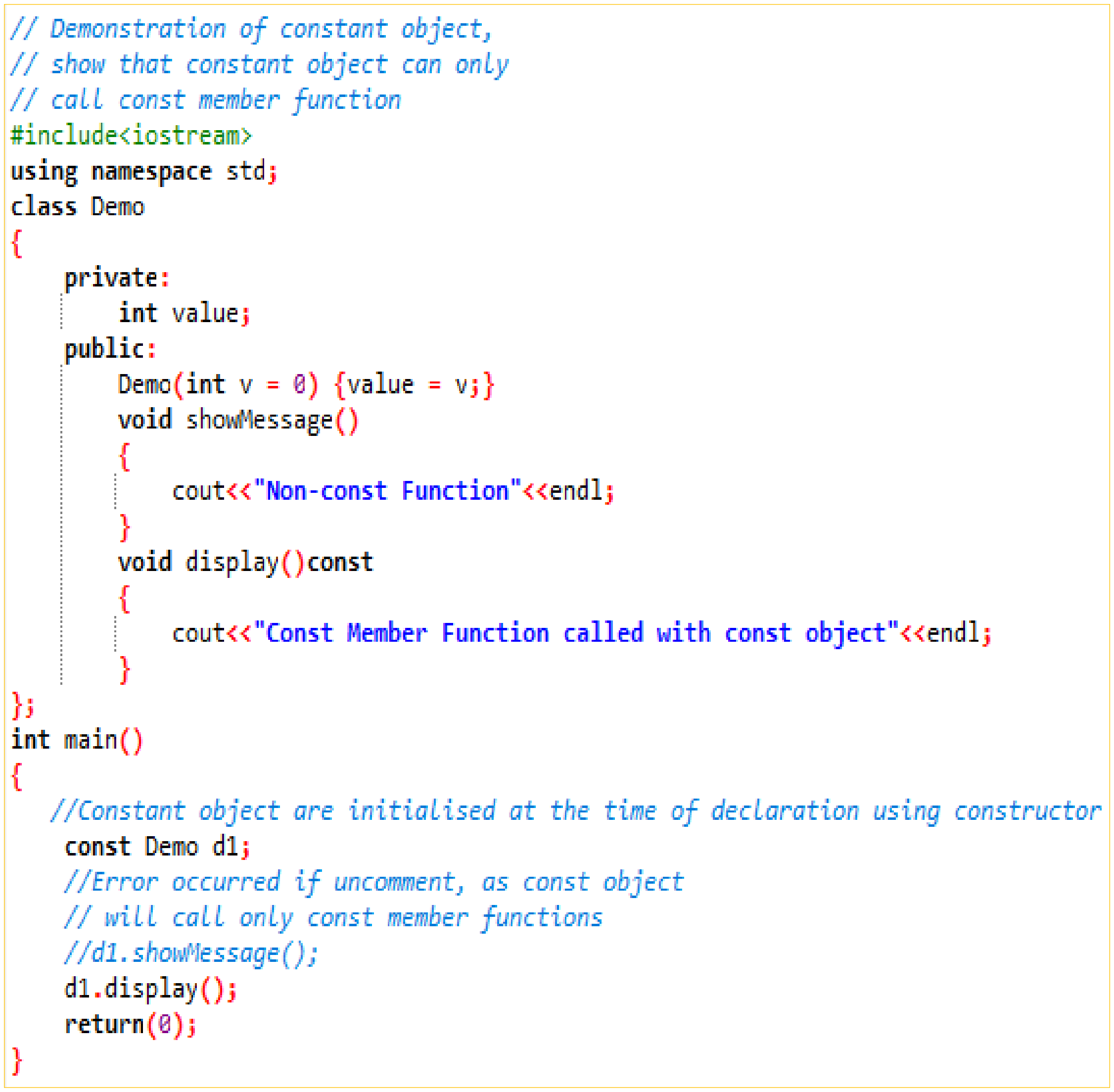
Output:



```
The value using object d: 28
The value using object d1 : 8
--------------------------------
```

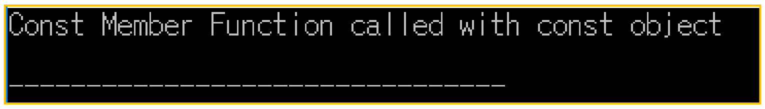
3. Constant Objects:

Like member functions and data members, the objects of a class **can also be declared as const**. An object declared as const **cannot** be **modified** and hence, can **invoke only const member functions** as these functions ensure **not to modify** the object. A const object **can be created by prefixing the const keyword** to the object declaration. Any attempt to change the data member of const objects results in a **compile-time error**. The following **example** illustrates the declaration and usage of constant object along with constant member function:



```
// Demonstration of constant object,
// show that constant object can only
// call const member function
#include<iostream>
using namespace std;
class Demo
{
    private:
        int value;
    public:
        Demo(int v = 0) {value = v;}
        void showMessage()
        {
            cout<<"Non-const Function"<<endl;
        }
        void display()const
        {
            cout<<"Const Member Function called with const object"<<endl;
        }
};
int main()
{
   //Constant object are initialised at the time of declaration using constructor
    const Demo d1;
    //Error occurred if uncomment, as const object
    // will call only const member functions
    //d1.showMessage();
    d1.display();
    return(0);
}
```

Output:



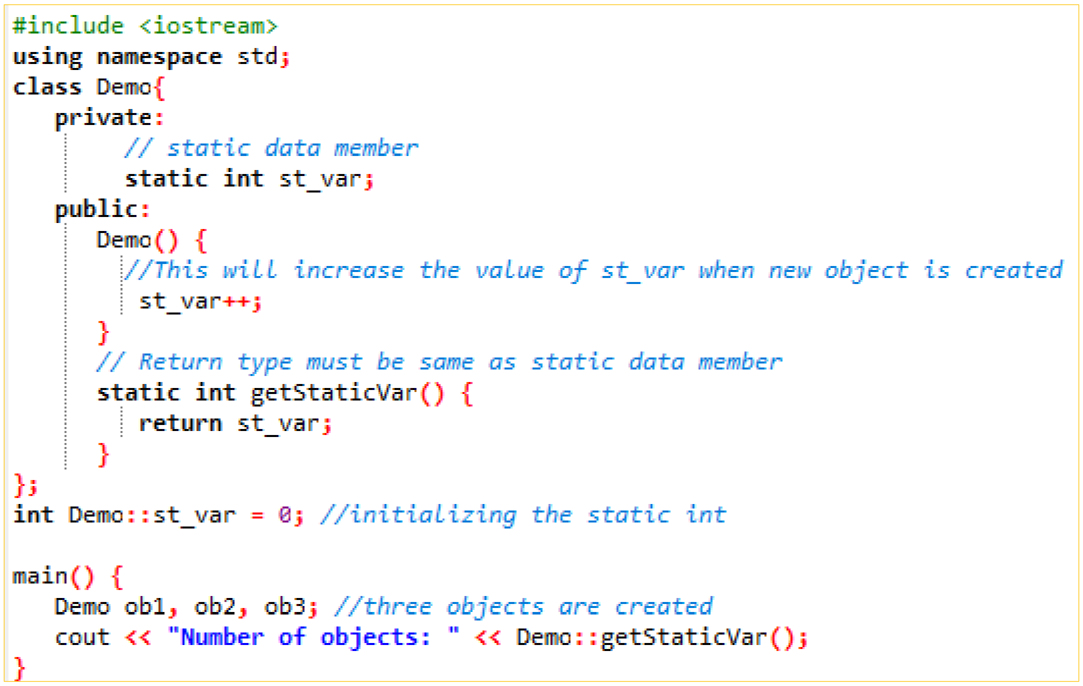
```
Const Member Function called with const object

-------------------------------
```

4. Static Data Members:

When a member variable is declared with the **keyword** **static**, there will be only **one copy of the member variable in memory**, **regardless** of the number of instances of the class that might exist. A single copy of a class's static member variable is **shared by all instances** of the class.

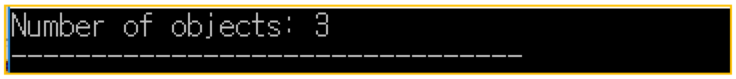
4.1 Accessing and Initialization

The following **example** illustrates the initialization and accessing process of static data members:



```
#include <iostream>
using namespace std;
class Demo{
   private:
        // static data member
        static int st_var;
   public:
      Demo() {
        //This will increase the value of st_var when new object is created
         st_var++;
      }
      // Return type must be same as static data member
      static int getStaticVar() {
         return st_var;
      }
};
int Demo::st_var = 0; //initializing the static int

main() {
   Demo ob1, ob2, ob3; //three objects are created
   cout << "Number of objects: " << Demo::getStaticVar();
}
```

Output:



```
Number of objects: 3
--------------------------------
```

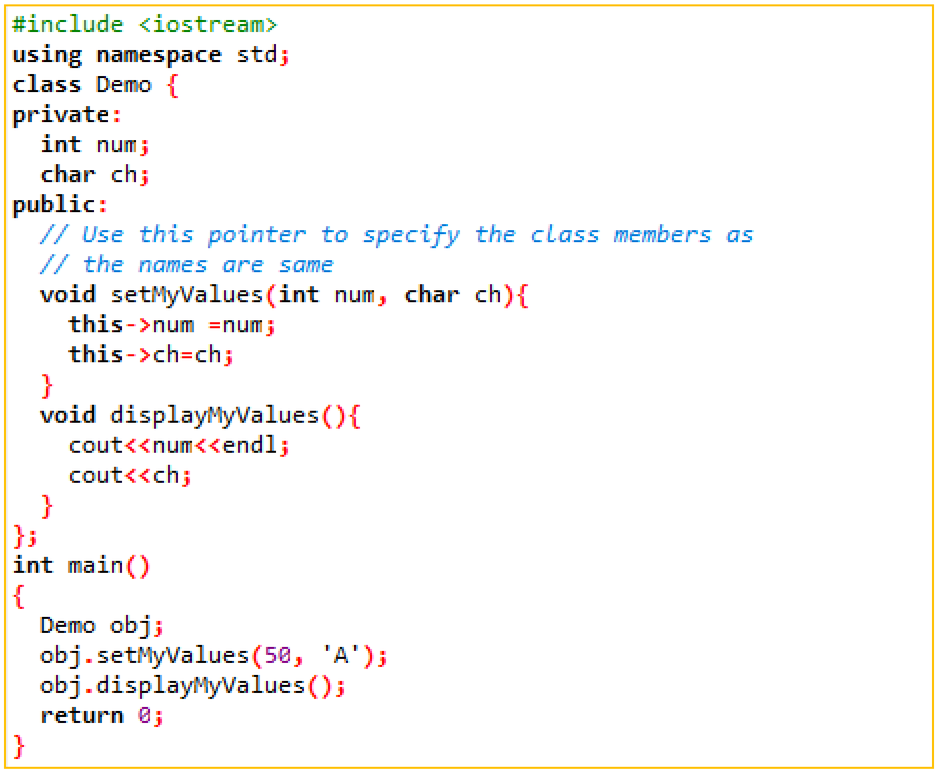
4.2 Lifetime

Even though static member variables are declared in a class, they are actually defined outside the class declaration. The **lifetime of a class's static member variable is the lifetime of the program**. This means that a class's static member variables come into existence before any instances of the class are created.

5. "this" Pointer:

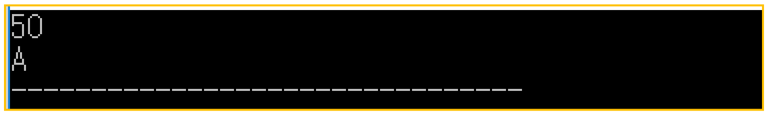
Every object has access to its own **address** through an important pointer called **"this"** pointer. The **"this"** pointer is an **implicit** parameter to all member functions. Therefore, inside a member function, this may be **used to refer to the invoking object**. To **summarize**, the **"this"** pointer holds the address of current object, in simple words you can say that this pointer points to the current object of the class.

Let’s take an example to understand this concept. Here, you can see that we have **two data members** num and ch. In member function **setMyValues()** we have two local variables having same name as data members name. In such case if you want to assign the local variable value to the data members then you won’t be able to do until unless you use this pointer, because the **compiler won’t know** that you are referring to object’s data members **unless** you use **“this”** pointer. This is one of the example where you must use **this** pointer.



```
#include <iostream>
using namespace std;
class Demo {
private:
  int num;
  char ch;
public:
  // Use this pointer to specify the class members as
  // the names are same
  void setMyValues(int num, char ch){
    this->num =num;
    this->ch=ch;
  }
  void displayMyValues(){
    cout<<num<<endl;
    cout<<ch;
  }
};
int main()
{
  Demo obj;
  obj.setMyValues(50, 'A');
  obj.displayMyValues();
  return 0;
}
```

Output:



```
50
A
--------------------------------
```

Lab Tasks

1. Create a Student class, where attributes associated with each student are name, registration number, father name, degree and department. All attributes should not be accessed directly. One can view the details of all students. Note student attributes can not be changed by any means after initialization. (Hint use constant objects)
2. Your team is creating a game ludo. You are working on player class. It contains the following information.
   - Current position of player (between 1 to 100)
   - Player alive pawns (Gooti)
   - Does the player have its turn now or not.

   Player data can be initialized via constructor and via one setter function. Player data (attributes) can be fetched using getter() function but with the conformation that via getting data player data cant be changed or updated. (user constant functions)
3. In student class (Question 1) maintaining student count means your program could be able to tell the count of student objects created. (User static counter variable)
4. You are creating an Employee Record App. Following information is required to store.
   - **name**. A string that holds the employee's name.
   - **idNumber**. An int variable that holds the employee's ID number.
   - **department**. A string that holds the name of the department where the employee works.
   - **position**. A string that holds the employee's job title.
   - **CovidStatus**: Either the employee has been covid Positive or not.

   Applications should have the following features.
   - Any kind of getter function or output function should not allow any kind of information or data change
   - Create a function that reports the total employee stored so far.
   - Add 2 employees' data with the surety that their data could never be changed or updated.
   - Employee ID number should be constant; it is not going to change ever.