# CL-1004 Object Oriented Programming

# LAB - 05

## This pointer, static and  Constant keywords, Array of Objects, Has a relation

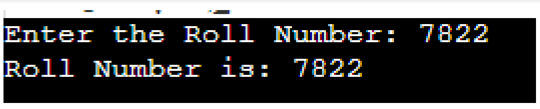
NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
Spring 2024

## This Keyword

This Keyword This keyword is a pointer to the current object. It is used to refer to the member variables and member functions of the current object. It can be particularly useful when there is a naming conflict between a member variable and a local variable or parameter with the same name in a member function. By using this, you can disambiguate between the two and access the member variable specifically.

**See below example:**

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

class MyClass{
   private:
      int roll_number;
   public:
      void set_roll_number(int roll_number){
         this->roll_number = roll_number;
      }
      void printData(){
         cout<<"Roll Number is : "<<roll_number;
      }
};

int main()
{
   MyClass my_class;
   int roll_num;
   cout<<"Enter the Roll Number : ";
   cin>>roll_num;
   my_class.set_roll_number(roll_num);
   my_class.printData();
}
```



```
Enter the Roll Number: 7822
Roll Number is: 7822
```

In this example, the MyClass class has a private member variable **roll_number**. The **set_ roll_number** member function takes an integer parameter **roll_number** and sets the value of the member variable to it using the this keyword. The **printData** member function simply prints the value of the member variable to the console, again using the this keyword to refer to the member variable.

**What will happen if we will not use “this pointer” in above example?**

**Which value will be assigned to member variable roll_number? And why?**
**Which will will be assigned to local variable roll_number? And why?**
**Before moving the next topic students must know the answers of above questions.**

## Static Keyword

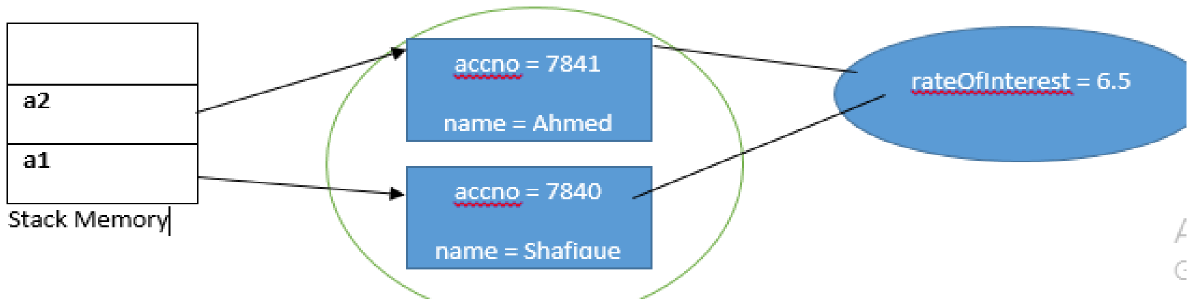
Static in C++ OOP is used to declare class-level variables and member functions **that are shared by all instances of the class.** It means that the variable is initialized once and remains in memory throughout the execution of the program, and that the member function can be called directly on the class, rather than on an instance of the class.

Let justify the statement **"Static variables are shared by all instances of the class"** by code.

```
#include<iostream>

using namespace std;

class Account{

   public:

   int accno; // data member also instance variable

   string name; // data member also instance variable

   static float rateofInterest; // static variable and

   //will be shared in qall instances of class.

      Account(int accno, string name)

      {

         this->accno = accno;

         this->name = name;


      }

      void Display(){

         cout<<accno<<" "<<name<<" "<<rateofInterest<<endl;

      }

};

float Account::rateofInterest=6.5;

int main(void)

{
   Account a1(7840,"Shafique");

   Account a2(7841, "Ahmed");

   a1.Display();

   a2.Display();

   return 0;

}
```

In above example,  **rateOfInterest**  is static which is shared between two instances( a1 and a2) of the Account class. There is only one copy of static variable **rateOfInterest** created in the memory. It is shared to all the objects.

**Memory mapping of above code**.



**Const Kewword:**

In C++ OOP, const is a keyword used to make a variable or function "unchangeable" once it has been defined. It helps prevent accidental changes to the value of a variable or the state of an object, making your code more reliable and easier to understand.

```
#include<iostream>

using namespace std;

class Circle{

    public:

    const double PI= 3.141;

    double radius;


    Circle(double radius){

        this->radius = radius;

    }


    double getArea() const // constant member function
    {
        return PI*radius*radius;
    }


};
int main(void)
{
    const Circle c(5); //constant object
    //c.radius=10; // attempt to modify c's radius (this will results in compile time error)
    cout<<"Area: " <<c.getArea()<<endl;
    return 0;
}
```

In The above example, the const keyword is used to create a constant variable PI, which cannot be modified after initialization. The const keyword is also used to declare a constant member function getArea(), which promises not to modify any non-static data members of the class In Main c object is declared as const. When you declare an object as const, it means that the object cannot be modified.

**Array of Objects:**

In C++, you can create arrays of objects just like you create arrays of primitive data types. Arrays of objects allow you to store multiple objects of the same class in contiguous memory locations.

**Here's a simple example demonstrating how to create an array of objects in C++:**

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


class MyClass{
  public:
  int num;
   //contructor to initialize num
  MyClass()
  {
   num=0;
  }
```

```
 //constructor with parameters
 MyClass(int n)
 {
  num=n;
 }
  // Method to display num
 void display()
 {
  cout<<"Number : "<< num <<endl;
 }
};

int main()
{
   const int size = 5; // Size of the array
   MyClass arr[size]; // Array of MyClass objects
   //Intializing objects in the array
   for (int i = 0; i < size; i++)
   {
      arr[i]=MyClass(i+1);
   }
   //Displaying objects in the array
   for (int i = 0; i < size; i++)
   {
      arr[i].display();//Displaying the number of each object
   }

   return 0;
```

```
}
```

**In this example:**

We define a class MyClass with a member variable num.
There are two constructors: one default constructor which initializes num to 0, and another constructor that takes an integer parameter and sets num to the value of the parameter.
The display() method prints the value of num for an object.
In the main() function, we create an array arr of MyClass objects with a size of 5 (size).
We initialize each object in the array with a number from 1 to 5 using a for loop.
Finally, we iterate through the array and call the display() method for each object to print its number.
This demonstrates how you can use arrays of objects in C++ to manage multiple objects of the same class efficiently.

**Example 2:**

```
#include<iostream>
#include<cstring>// for strcpy()
using namespace std;
class Employee{
   int id;
   char name[30];
   public:
   //Declaration of function
   void take_data(int id, char name[]);
   void display_data();
};
//Defining the function outside the class
void Employee::take_data(int id, char name[])
{
   this->id = id;
   strcpy(this->name, name);

}
//Defining the function outside the class
void Employee::display_data()
{
   cout<< id <<" "<<name<< " " <<endl;
}

int main()
{
   //This is an array of objects having maximum limit of 30 Employees

   Employee emp[30];
   int n, i;
   cout<<"Enter Number of Employees - ";
   cin>>n;

   //Accessing the function

   for(i=0; i<n; i++)
```

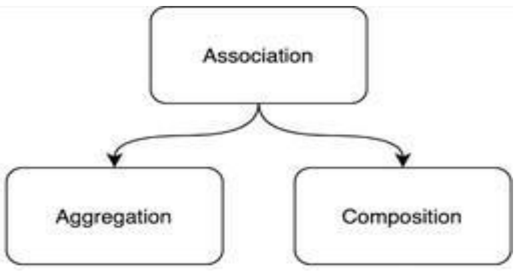
```
  {
     int id;
     char name[30];
     cout<<"Enter Id : ";
     cin>>id;
     cout<<"Enter Name : ";
     cin>>name;
     emp[i].take_data(id, name);
  }
cout<<"Employee Data - " <<endl;
//Accessing the Function
for ( i = 0; i < n; i++)
{
  emp[i].display_data();
}

}
```

**HAS-A RELATIONSHIP**

In Object-Oriented Programming (OOP), a has-a relationship is a type of association between classes where one class has a member variable of another class type.

This relationship is also known as composition or aggregation.

Association, Aggregation, and Composition are about Has a relationship.

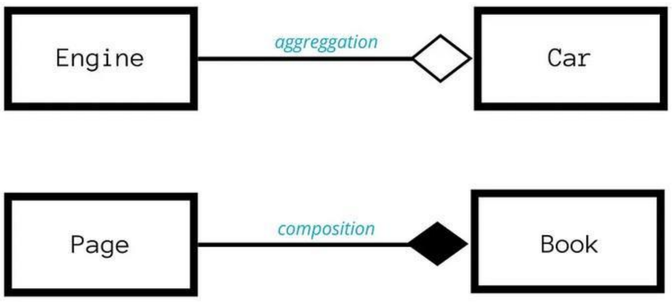


**Association**

A type of relationship between classes in C++ called association explains how objects of one class are connected to objects of another class. An instance of association is one in which one class makes use of or interacts in some manner with another class. Aggregation and Composition are subsets of Association that describe relationships more accurately.

Aggregation - **independent** relationship. An object can be passed and saved inside the class via a constructor, method, or setter.

Composition - **dependent** relationship. An object is **created** by the owner object



- Car uses Engine -> this is an aggregation

- Book needs Pages -> is composition.

**Aggregation**: This is a type of association where objects from different classes are referenced but can still live separately.

Due to the fact that it has a pointer to an Engine object, the Car class in this example has an aggregate relationship with the Engine class. Several Car objects may share the same Engine object, and the Engine object may exist separately from the Car object.

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

class Engine {
public:
   string model;
   int year;

   Engine(string model, int year) {
      this->model = model;
      this->year = year;
   }
};

class Car {
   Engine *e;
public:
   int id;
   string model;

   Car(int id, string model, Engine *e) {
      this->id = id;
      this->model = model;
      this->e = e;
   }

   void display() {
      cout << id << " " << model << " " << e->model << " " << e->year
 << endl;
   }
};

int main() {
   Engine el("HAJ", 2018);
   Car c1(101, "Toyota", &el);
   c1.display();
   return 0;
}
```

**Composition**

This form of association involves the composition of two classes, in which case the containing object determines the lifetime of the composed object. In other terms, the composed object is also destroyed when the containing object is.

In this example, the Book class contains an array of one hundred Page objects, each of which is created using the default constructor of the Page class. The pages cannot exist on their own, when the object of a book is destroyed so is the Page object.

```
 Class page{

        //implementation of page class
  };
Class Book{

        Page pages[100];  array of 100 pages
    Public:
     Book(){
            //create 100 pages for our array
                For(int i=0; i< 100 ; i++){
            pages[i]= page();
        }
            }
      //other member function of Book class

  };
```

## ***Lab 5 Exercise:***

**Task-01:**

Create a class named RealtorCommission. Fields include the sale price of a house, the sales commission rate, and the commission. Create two constructors. Each constructor requires the sales price (expressed as a double) and the commission rate. One constructor requires the commission rate to be a double, such as .06. The other requires the sale price and the commission rate expressed as a whole number, such as 6. Each constructor calculates the commission value based on the price of the house multiplied by the commission rate. The difference is that the constructor that accepts the whole number must convert it to a percentage by dividing by 100. Also include a display function for the fields contained in the RealtorCommission class.
**Write a main()function that instantiates at least two RealtorCommission objects—one that uses a decimal and one that uses a whole number as the commission rate**

**.**

**Task-02:**
Let's consider a car rental system instead of a hotel rent calculator.

1. The car rental system requires a module that can calculate the rental charges for customers. The system should allow for independent changes in implementation while keeping the interface intact, enabling flexibility for evolving requirements.
2. The rental charges for each customer are $50.75 per day, a rate set by the car rental company's committee and subject to complex formalities for change.
3. The module should accept the customer's name and the number of rental days as arguments in the constructor. The customer's name must be initialized only once during construction, and any subsequent attempts to change it should fail.
4. Based on the number of rental days, the module determines if the customer is eligible for a discount. Customers renting for more than a week are entitled to a discount, otherwise, they are charged at the standard rate.
5. The discounted rental fee is calculated after deducting one day from the total rental period.
6. At the end, the module displays the following details:
   a. Customer Name
   b. Rental Days
   c. Rental Amount

The display function should be read-only and must not modify any data member.

**RentalCalculator:**
**-rentPerDay**
**- customerName**
**- numberOfDays**
**- customerRent**
**Methods:**
**+ RentCalculator();**
**+ RentWithDiscount();**
**+ RentWithoutDiscount();**
**+ DisplayRent();**

**Instructions:**
**Implement the class structure as specified. Use appropriate data types, return types, and function arguments. Demonstrate the results for two initialized instances.**

**Task-03:**

User Construct a class named Circle that has a floating-point data member named radius getting from user input. The class should have a zero-argument constructor that initializes this data member to 0. It should have member functions named calcCircumference() and calcArea() that calculate the circumference and area of a circle respectively, a member function setRadius()to set the radius of the circle, a member function getRadius() to return the radius, and a member function showData() that displays the circle's radius, circumference, and area. The formula for the area of a circle is A=πr2. The formula for the circumference of a circle is C=2πr.
Note: Without using this <cmath>

**Task-04:**
Let's consider a bookstore inventory system instead of a restaurant ordering system:

a) Books hold two things: title and price.
b) Use getter and setter function for both variables.
c) Inventory is a class that holds an array of books. You can have different functions to add and remove items, or display the entire inventory.
d) The bookstore inventory system has one inventory.
e) Any staff member can place an order to the system on behalf of a customer. The order class consists of one or more books and a payment.
f) e) Payment is a class that holds the amount of money that a customer needs to pay

This is generated when the order is placed. In this scenario: The entities are: Books, Inventory, Order, and Payment. The relationships are: Inventory holds Books. Order consists of one or more Books and a Payment. Payment is associated with an Order.

**Task-05:**
Implement a library management system that holds class information about the library's books, patrons, loans, and fines. Identify the relationships between the four entities mentioned. Keep in mind the following information:

- Books have attributes such as title, author, ISBN, and genre.
- Create Parametrize constructor for Book Class.
- Patrons have attributes such as Name, Barrowerbook.
- Patrons can borrow multiple books from the library.
- Create Parametrize constructor for Patrons Class.
- Each book can be borrowed by one patron at a time.
- Loans track which patron has borrowed which book and the due date for returning the book.

Your task is to design classes and establish relationships between them to create a functional library management system.