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 2025

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();
   return 0;
}
```

```
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

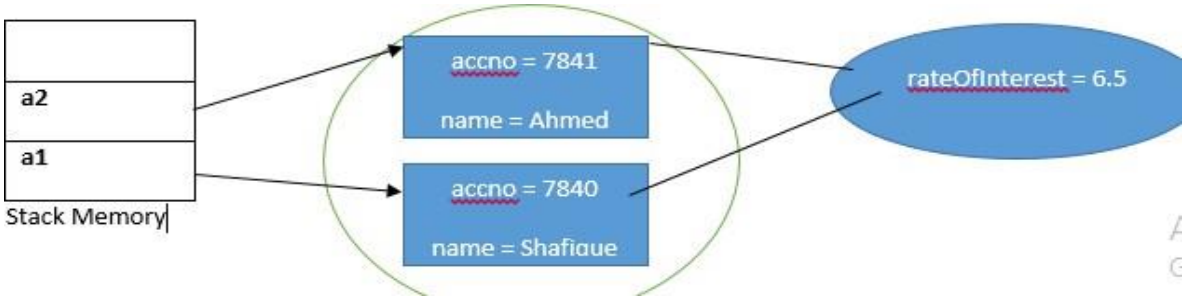
Static in C++ OOP is used to declare class-level variables and member functions **that are shared by all instances of the class.** It means that the variable is initialized once and remains in memory throughout the execution of the program, and that the member function can be called directly on the class, 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 varibale and
       //will be shared in all insrances of objects;
       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 =Account(7840, "Shafique");
    Account a2=Account(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 Keyword:

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 = 3.14159; // constant variable
       double radius;

       Circle(double radius)
        {
            this->radius = radius;
        }
       double getArea() const // const memeber 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 result 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++:

```cpp
#include <iostream>
using namespace std;
class MyClass {
public:
    int num;
    // Constructor to initialize num
    MyClass() {
        num = 0;
    }

    // Constructor with parameter
    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
    // Initializing objects in the array
    for (int i = 0; i < size; i++) {
        arr[i] = MyClass(i + 1);} // Initializing with numbers 1 through 5
    // 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:

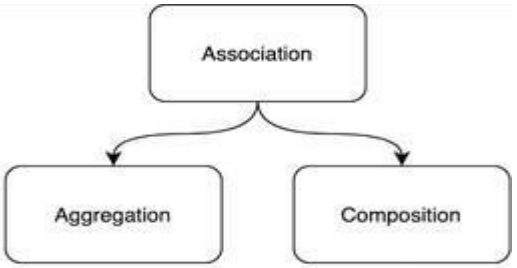
```cpp
#include<iostream>
#include<cstring> // for strcpy()
using namespace std;
class Employee
{
int id;
char name[30];
public:
// Declaration of function
void take_data(int id, char name[]);
// Declaration of function
void display_data();
};
// Defining the function outside the class
void Employee::take_data(int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

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 Student {
public:
   string name;
   Student(string n) : name(n) {}
   void show() { cout << "Student: " << name << endl; }
};

class University {
public:
   string name;
   Student* student;  // Aggregation: University has a reference to Student

   University(string n, Student* s) : name(n), student(s) {}

   void show() {
      cout << "University: " << name << endl;
      student->show();
   }
};

int main() {
   Student s1("Ali");  // Student exists independently
   University u1("Fast", &s1);
   u1.show();

   cout << "University object destroyed, but student still exists!\n";
   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.

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

class Engine {
public:
   Engine() { cout << "Engine Created\n"; }
   void start() { cout << "Engine Started\n"; }
   ~Engine() { cout << "Engine Destroyed\n"; }
};

class Car {
public:
   Engine engine;  // Composition: Car owns Engine

   Car() { cout << "Car Created\n"; }
   void start() { engine.start(); }
   ~Car() { cout << "Car Destroyed\n"; }
};

int main() {
   {
      Car myCar;
      myCar.start();
   } // myCar goes out of scope, so Car and Engine are destroyed.

   cout << "Car object no longer exists!\n";
   return 0;
}
```