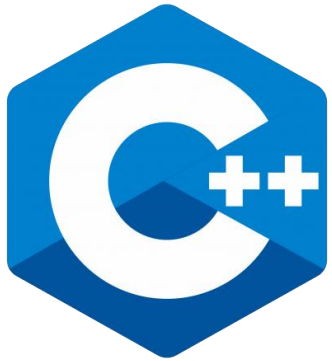


C++ Programming

Trainer : Pradnyaa S. Dindorkar

Email: pradnya@sunbeaminfo.com

1. References
2. Difference between Pointers and reference
3. Sum function and Copy Constructor
4. New and delete
5. Difference between New and malloc
6. Shallow Copy and deep copy

Todays topics

1. static
2. Friend function
3. Operator Overloading
4. Object Oriented programming structure(oops)
5. Major pillars of oops
6. Minor pillars of oops
7. Association

Static Variable

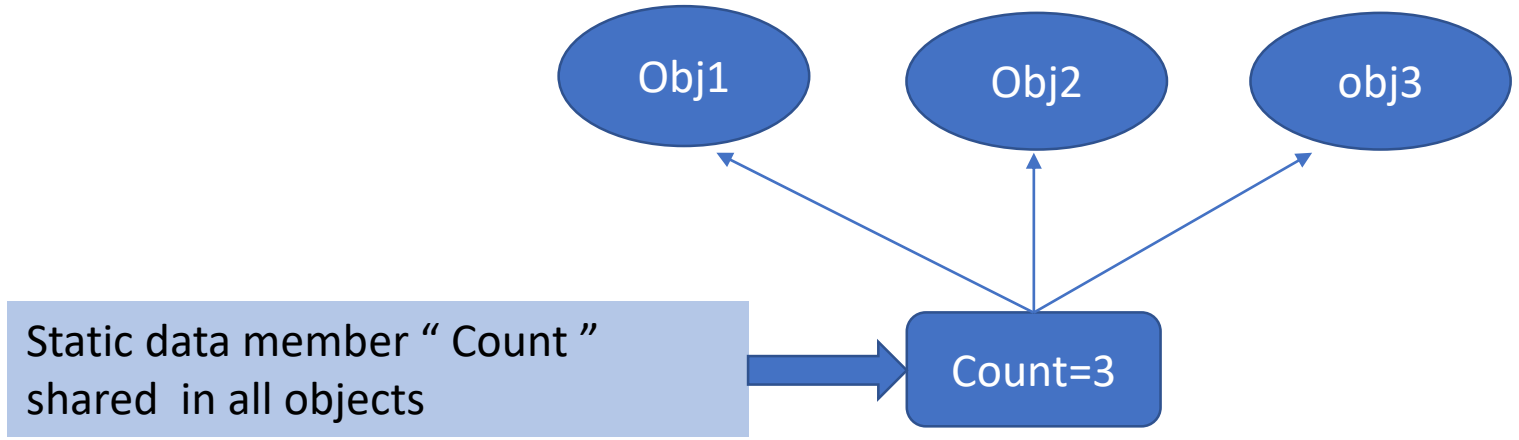
- All the static and global variables get space only once during program loading / before starting execution of main function
- Static variable is also called as shared variable.
- Initialized static and global variable get space on Data segment.
- Default value of static and global variable is zero.
- Static variables are same as global variables but it is having limited scope.

Static Data member

- If we want to share value of data member between all objects of same class then we should declare that data member as static data member.

- It is mandatory to provide global definition of static data member otherwise linker generates error.

- Static data member get space during class loading per class so it is called as **class-level variable**



Static Member Function

- Except main function, we can declare global function as well as member function static.
- static members of the class we should declare member function **static**.
- static member function is also called as **class level method.**
- To access class level method we should use classname and ::(scope resolution) operator.
- This pointer is not available in static member function .

Friend :-

- A non member function of a class which designed to access private data of a class is called friend function.

- To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend**

```
class MyData
{
private:
        int pin;
        int pass;

public:
        MyData(int pin,int pass);
        void PrintMyAccDetails();
        friend void anyFunction();
};
```

```
void anyFunction()
{
        MyData d1;
        d1.pass=9898;
        d1.pin=9999;
        d1.PrintMyAccDetails();
}
```

C++ is not a pure Object Oriented Language Because

- You can write code without creating a class in C++, and main() is a global function.
- Support primitive data types, e.g., int, char, etc. Instances(variable) of these primitive types are NOT objects.
- C++ provides "Friend" which is absolute corruption to the OO-Principle of encapsulation.

Operator Overloading

- operator is token in C/C++. And it is used to generate expression.
- Types of operator:
  - Unary operator (++,--,&,!,~,sizeof())
  - Binary Operator (Arithmetic, relational, logical , bitwise, assignment)
  - Ternary operator (conditional)
- In C++, also we can not use operator with objects of user defined type directly.
- If we want to use operator with objects of user defined type then we should **overload operator**.
- To overload operator, we should define **operator function.**
- **We can define operator function using 2 ways:**
  - **Using member function**
  - **Using non member function**

Operator Overloading

using member function

- operator function must be member function
- If we want to overload, binary operator using member function then operator function should take only one parameter.
  - Example :

```
c3 = c1 + c2;   //will be called as
                //c3 = c1.operator+( c2 )
```

using non member function

- Operator function must be global function
- If we want to overload binary operator using non member function then operator function should take two parameters.
  - Example :

```
c3 = c1 - c2;  // will be called as
               // c3 = operator-(c1,c2);
```

We can not overloading following operator using member as well as non member function:

1. dot/member selection operator(.)
2. Pointer to member selection operator(.*)
3. Scope resolution operator(::)
4. Ternary/conditional operator(? :)
5. sizeof() operator
6. typeid() operator
7. static_cast operator
8. dynamic_cast operator
9. const_cast operator
10. reinterpret_cast operator

We can not overload following operators using non member function:

- Assignment operator(=)
- Subscript / Index operator([])
- Function Call operator[()]
- Arrow / Dereferencing operator(→)

Object Oriented programming structure(oops) :-

-> It is a programing methodology to organise complex program into simple program in terms of class and objects such methodology is called as "Object Oriented programming structure"

-> It is a programing methodology to organise complex program into simple program by using the concept of Abstraction, Encapsulation
and Inheritance, modularity.

->so the language which supports Abstraction, Encapsulation and Inheritance is called as Object Oriented programming language.

Major pillars of oops

- **Abstraction**
  - getting only essential things and hiding unnecessary details is called as abstraction.
  - Abstraction always describe outer behavior of object.
  - In console application when we give call to function in to the main function , it represents the abstraction.
  - By Creating object and calling public member function on it we can achieve abstraction.
- **Encapsulation**
  - binding of data and code together is called as encapsulation. By defining class we can achieve encapsulation.
  - Implementation of abstraction is called encapsulation.
  - Encapsulation always describe inner behavior of object
  - Function call is abstraction and Function definition is encapsulation.
  - Information hiding
    - Hiding information from user is called information hiding.
    - In c++ we used access Specifier to provide information hiding.
- **Modularity**
  - Dividing programs into small modules for the purpose of simplicity is called modularity.
- **Hierarchy**
  - Hierarchy is ranking or ordering of abstractions.
  - Main purpose of hierarchy is to achieve re-usability.
  - Types –> 1: **Inheritance [is-a] , 2: Association [has-a]**

Minor pillars of oops

- **Polymorphism (Typing)**
  - One interface having multiple forms is called as polymorphism.
  - Polymorphism have two types
    1. **Compile time polymorphism (**Static polymorphism / Static binding / Early binding / Weak typing / False Polymorphism **)**

       when the call to the function resolved at compile time it is called as compile time polymorphism. And it is achieved by using function overloading, operator overloading, template
    2. **Runtime polymorphism (**Dynamic polymorphism / Dynamic binding / Late binding / Strong typing / True polymorphism**)**

       when the call to the function resolved at run time it is called as run time polymorphism. And it is achieved by using function overriding.
- **Concurrency**
  - Process of executing multiple tasks simultaneously is called Concurrency**.**
  - Can be achieved by multithreading which is Used to utilize hardware resources efficiently.
- **Persistence**
  - Used to maintain state of object across time and space on secondary storage .
  - Using file handling we can achieve it. To transfer and save the state of object needs serialization and also socket programming for network.

Association

- If has-a relationship exist between two types then we should use association.
- Example : Car has-a engine (OR engine is part-of car)
- If object is part-of / component of another object then it is called association.
- If we declare object of a class as a data member inside another class then it represents association.

Example Association:

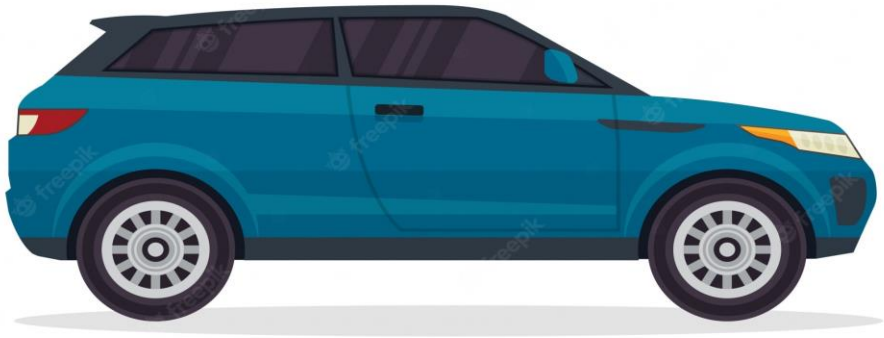
- Car has-a engine
- Laptop has-a hand disk
- Room has-a wall
- Bank has-a accounts

```
class Engine
{
        int cc, fuel;
};
class Car
{       private:
        Engine e;   //Association
};
```

Dependant Object : Car Object

Dependency Object : Engine Object

Example of Association



Composition and aggregation are specialized form of association

Composition

- If dependency object do not exist without Dependent object then it represents composition.
- Composition represents tight coupling.
- Example: Human has-a heart.

```
class Heart
{ };
class Human
{
   Heart hrt; //Association->Composition
};
```

Dependent Object : Human Object

Dependency Object : Heart Object

Aggregation

- If dependency object exist without Dependent object then it represents Aggregation.
- Aggregation represents loose coupling.
- Example: Department has-a Faculty.

```
class Faculty
{ };
class Department
{
   Faculty f; //Association->Aggregation
};
```

Dependent Object : Department Object

Dependency Object : Faculty Object

Tomorrows topics

- Inheritance
  - Protected Data member
  - Types of Inheritance
  - Mode of inheritance
  - Diamond Problem

Thank You