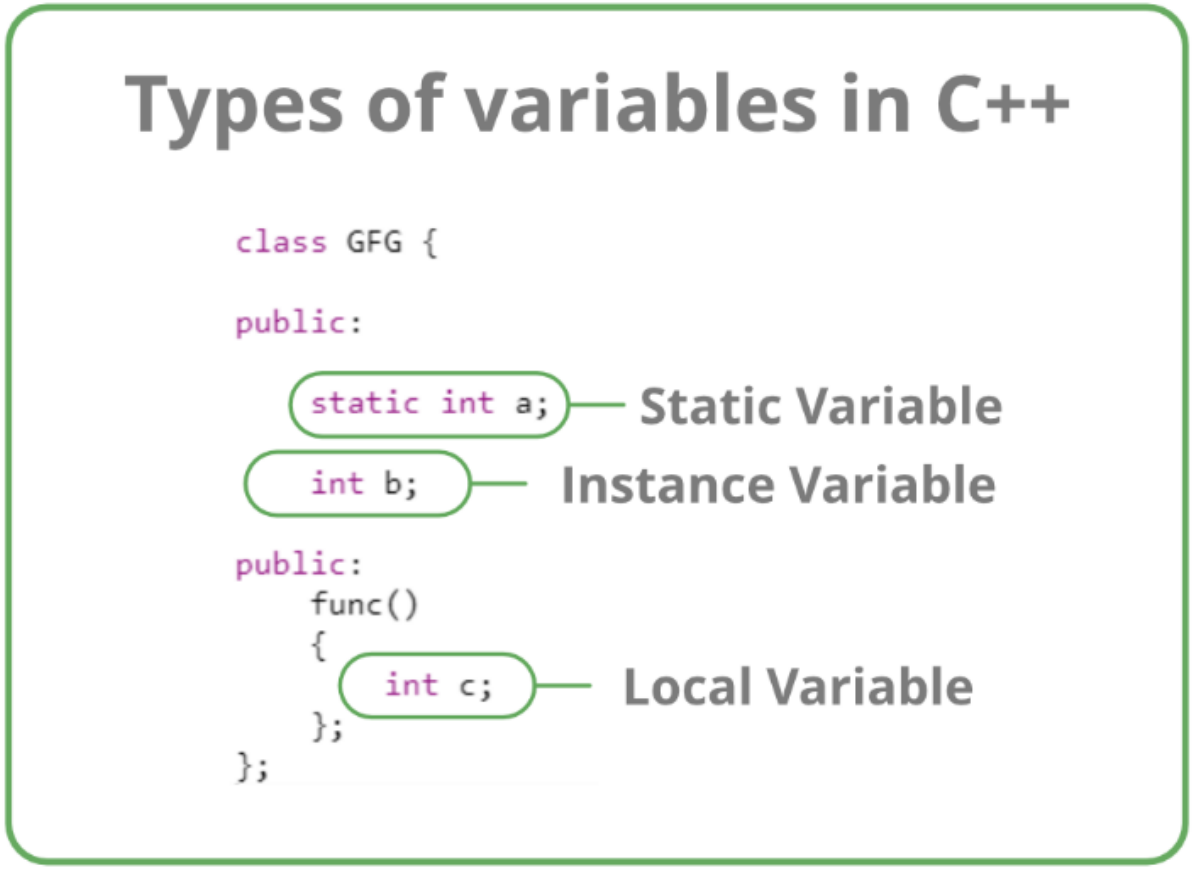
Variables & this Pointer in C++

Md. Alamgir Hossain
Senior Lecturer,
Dept. of CSE, Prime University
Mail: *alamgir.cse14.just@gmail.com*
YouTube: *https://www.youtube.com/alamgirhossaincse*
Blog: *https://alamgirhossainjust.blogspot.com/*

Types of Variables (Based on Scope of Variables)



Local Variable

- A variable defined within a block or method or constructor is called local variable.
- These variables are created when the ***block is entered or the function is called and destroyed after exiting from the block*** or when the call returns from the function.
- The scope of these variables exists ***only within the block in which the variable is declared.*** i.e. we can access these variables only within that block.
- ***Initialization of Local Variable is Mandatory.***

Instance Variable

- Instance ***variables are non-static variables and are declared in a class outside any method, constructor, or block.***
- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.
- ***Initialization of Instance Variable is not Mandatory.***
- Instance Variable can be accessed only by creating objects.

C++ Static Keyword

- In C++, ***static is a keyword or modifier*** that belongs to the type not instance.
- ***So, instance is not required to access the static members.***
- In C++, static can be field, method, constructor, class, properties, operator, and event.
- **All the data members in a *static function* must be static.**

Advantages of C++ Static Keyword

- **Memory efficiency:** Now we don't need to create an instance for accessing the static members, so it saves memory.
- Moreover, it belongs to the type, so ***it will not get memory each time when the instance is created.***
- A field which is declared as static is called static field. Unlike instance field which gets memory each time whenever you create object, there is ***only one copy of static field created in the memory.*** It is shared to all the objects.

Example of Different Types Variable

```cpp
#include<iostream>
using namespace std;
class Variable{
public:
    static int ID;//Static Variable
    string name = "ABCD";//Instance Variable
    void display()
    {
        string mobile = "234567";//Local Variable
        cout<<mobile<<endl;
    }
};
int Variable::ID = 1234;
int main()
{
    cout<<Variable::ID<<endl;//Accessing static variable
    Variable obj;
    cout<<obj.name<<endl;//Accessing Instance variable
    obj.display();
    return 0;
}
/*
========Output========
1234
ABCD
234567
*/
```

A general C++ Static Function

```
1  #include<iostream>
2  using namespace std;
3  void display()
4  {
5      static int num1;
6      int num2 = 0;
7      num1++;
8      num2++;
9      cout<<num1<<" "<<num2<<endl;
10 }
11 int main()
12 {
13     display();
14     display();
15     return 0;
16 }
```

Example of C++ Static Field with the Concept of OOP

```cpp
#include<iostream>
using namespace std;
class student{
private:
    static int id; double res;
public:
    student(int ID, double Res){
        id = ID; res = Res;
    }
    void display(){
        cout<<"ID: "<<id++<<", Result: "<<res<<endl;
    }
};
int student::id; //Definition of Static Variable
int main()
{
    student o1(100, 3.50);
    o1.display(); o1.display(); o1.display();

    student o2(200, 3.50); o2.display(); o2.display(); o2.display();
    return 0;
}
/*
======Output=====
ID: 100, Result: 3.5
ID: 101, Result: 3.5
ID: 102, Result: 3.5
ID: 200, Result: 3.5
ID: 201, Result: 3.5
ID: 202, Result: 3.5
*/
```

How Static Variable Allocate Same Memory location!

```cpp
class student{
private:
    static int id; double res;
public:
    student(int ID, double Res){
        id = ID; res = Res; }
    void display(){
        cout<<"ID: "<<id++<<", Result: "<<res<<endl;
        cout<<"Memory Address of id: "<<&id<<", and res: "<<&res<<endl;
    }
}; int student::id; //Definition of Static Variable
int main()
{
    student o1(100, 3.50); o1.display(); o1.display(); o1.display();
    student o2(200, 3.50); o2.display(); o2.display(); o2.display();
    return 0;
}
/*
======Output=====
ID: 100, Result: 3.5
Memory Address of id: 0x489008, and res: 0x69fee8
ID: 101, Result: 3.5
Memory Address of id: 0x489008, and res: 0x69fee8
ID: 102, Result: 3.5
Memory Address of id: 0x489008, and res: 0x69fee8
ID: 200, Result: 3.5
Memory Address of id: 0x489008, and res: 0x69fee0
ID: 201, Result: 3.5
Memory Address of id: 0x489008, and res: 0x69fee0
ID: 202, Result: 3.5
Memory Address of id: 0x489008, and res: 0x69fee0
*/
```

Value Initialization in C++ Static Variable

```cpp
#include<iostream>
using namespace std;
class p{
private:
    int a = 10;
    string s = "Prime";
    static float num;
public:
    void func()
    {
        num = 100;
    }
    void dispaly(){
        cout<<a<<" "<<s<<" "<<num++<<endl;
    }
};
float p::num;
int main()
{
    p obj, obj2, obj3;
    obj.func();
    obj.dispaly();
    obj2.dispaly();
    obj3.dispaly();
    return 0;
}
```

Static Member Function

```
1  #include<iostream>
2  using namespace std;
3  class Static_Learning{
4  private:
5      static int id;
6      static double result;
7  public:
8      static void Value_Assign(){//Used for assigning values into the static variables
9          id = 12345;
10         result = 3.50;
11         //All data members must be static inside the static member function
12     }
13     void display()
14     {
15         cout<<"Id: "<<id<<", Result: "<<result<<endl;
16     }
17 };
18 int Static_Learning::id;
19 double Static_Learning::result;
20 int main()
21 {
22     Static_Learning SL;
23     SL.Value_Assign();
24     SL.display();
25     return 0;
26 }//Output: Id: 12345, Result: 3.5
27
```

C++ this Pointer

In C++ programming, ***this is a keyword*** that refers to the current instance of the class. There can be 3 main usages of this keyword in C++.

- It can be used to ***pass the current object as a parameter to another method***.
- It can be used to ***refer current class instance variable***.
- It can be used to ***declare indexers.***

Example of C++ this Pointer

```cpp
#include <iostream>
using namespace std;
class Employee {
    public:
        int id; //data member (also instance variable)
        string name; //data member(also instance variable)
        float salary;
        Employee(int id, string name, float salary)
         {
              this->id = id;
              this->name = name;
              this->salary = salary;
         }
        void display()
         {
             cout<<id<<"  "<<name<<"  "<<salary<<endl;
         }
};
int main(void) {
    Employee e1 = Employee(101, "Sopno", 890000); //creating an object of Employee
    Employee e2 = Employee(102, "Sadhin", 59000); //creating an object of Employee
    e1.display();
    e2.display();
    return 0;
}
```

Thank You