F.Y.B.SC.IT – SEM II

OBJECT ORIENTED PROGRAMMING WITH C++ (PUSIT206T)

By,

Prof. Nikita Madwal

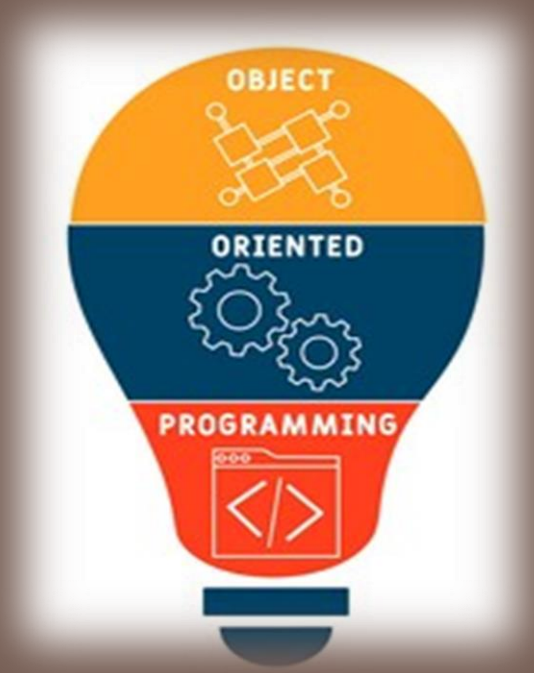
UNIT 1

1. INTRODUCTION OF OBJECT-ORIENTED DESIGN

2. STARTING WITH C++

3. FEATURES OF C++

4. OPERATORS AND REFERENCES IN C++



4. OPERATORS AND REFERENCES IN C++

Introduction

- C++ has a rich set of operators.
- All C operators are valid in C++ also.
- In addition, C++ introduces some new operators.
- This chapter discuss all the new operators introduced by C++.

Scope Resolution Operator

- The operator **::** is known as **scope resolution operator** in C++.
- The operator is used for two purposes :
- PURPOSE-1 : For accessing global variables.
- PURPOSE-2 : Identifying class members to which class they belong.

- **<u>PURPOSE-1 : For accessing global variables.</u>**
- In C++ block can be created by using {and}.
- Any variable declared within that block is confine within that block only.
- Now **if a variable declared within any block and a global variable having same name, the priority is given to the local variable.**
- What if **we want to use the global variable having the same name in the block too ? <u>Scope resolution operator helps in these situations</u>.**

- E.g.

scop1.cpp

```
1   #include<iostream>
2   using namespace std;
3
4   int x=40; // Global x
5
6   int main()
7   {
8   int x = 10; // Local x
9   cout << "Value of global x is " << ::x;
10  cout << "\nValue of local x is " << x;
11  return 0;
12  }
```

- Output:

```
C:\Users\Admin\Desktop\C++Practical\scop1.exe
Value of global x is 40
Value of local x is 10
--------------------------------
Process exited after 0.01901 seconds with return value 0
Press any key to continue . . .
```

- **<u>PURPOSE-2 : Identifying class members to which class they belong</u>**
- In this, **it is useful to** keep only the **function declarations within the class specifier and define the functions outside it.**
- To tell the compiler that a function is a member of a class it has to be declared within the specifier or it can be defined outside the class specifier **using a scope resolution operator '::'** (double colon symbol) with the function definition.

E.g.

outsidefunction.cpp

```cpp
#include<iostream>
using namespace std;
class Car
{
  private:
    int car_number;
    char car_model[10];
  public:
    void getdata();                    //function declaration
    void showdata();
};
void Car::getdata()                    // function definition
 {
   cout<<"Enter car number: ";
   cin>>car_number;
   cout<<"Enter car model: ";
   cin>>car_model;
 }
void Car::showdata()
 {
   cout<<"Car number is "<<car_number<<endl;
   cout<<"Car model is "<<car_model;
 }
// main function starts
int main()
 {
    Car c1;
    c1.getdata();
    c1.showdata();
    return 0;
 }
```

- Output:

F:\PATKAR_BSCIT\F.Y.B.Sc.IT\OOP's\NIKITA_OOPs\Practical_Related_Data\Practicals\outsidefunction.exe

```
Enter car number: 9090
Enter car model: Audi
Car number is 9090
Car model is Audi
--------------------------------
Process exited after 8.137 seconds with return value 0
Press any key to continue . . .
```

Reference Variables

- A reference variable **is a variable** which can **take reference of a previous defined variable or any constant.**
- When a **variable is declared as a reference,** it becomes an **alternative name for an existing variable.**
- A variable can be declared as a reference **by putting '&' in the declaration.**
- The general syntax for defining a reference variable is

data_type &reference_name = referent;

NOTE : A reference variable must be initialized when it is declared i.e., if you write int & x; and

later write x = y will be invalid.

- E.g.

reference.cpp

```cpp
#include<iostream>
using namespace std;
int main()
{
int x=10;
int &refx=x;
cout<<"x= "<<x<<endl;

refx++;
cout<<"refx= "<<refx<<endl;
return 0;
}
```

- Output:

```
C:\Users\Admin\Desktop\C++Practical\reference.exe

x= 10
refx= 11


--------------------------------
Process exited after 0.02041 seconds with return value 0
Press any key to continue . . .
```

The Bool Data Type

- The **bool data type** is a new data type in C++ which is **used with Boolean values.**
- We can create variables of type **bool type** which can **store any true or false value.**
- A Boolean data type is declared with the **bool keyword**
- Even we can **assign integer values** to bool type variables.
- Any **non zero value** is termed as **true** and any **0 value** is considered as **false.**
- Integer **value 1** is used to represent **true** and ***0*** for representing ***false*.**
- We can **also use keywords true and false** for **representing true and false value.**

E.g.

bool1.cpp

```cpp
#include <iostream>
using namespace std;
int main()
{
  int x1 = 10, x2 = 20,m=2;
  bool b1, b2;

  b1 = (x1 == x2);  //false
  b2 = (x1 < x2);    // true
  cout << "Bool1 is = " << b1 << "\n";
  cout << "Bool2 is = " << b2 << "\n";

  bool b3 = true;
  if(b3)
  {
    cout << "Yes" << "\n";
  }
  else
  {
    cout << "No" << "\n";
  }
  int x3 = false + 5 * m - b3;
  cout << x3;
  return 0;
}
```

- Output:

C:\Users\Admin\Desktop\C++Practical\bool1.exe

```
Bool1 is = 0
Bool2 is = 1
Yes
9
--------------------------------
Process exited after 0.01583 seconds with return value 0
Press any key to continue . . .
```

The Operator New and Delete

- An object is created when its definition in a program and it is destroyed when its name goes out of scope or the program terminates.
- In C++,the **operator new creates** such **objects** and the **operator delete** can be used **to destroy them** later.

The new operator

- The new operator **is used to create a memory space for an object** of a class.
- The new operator returns a pointer to the object created.
- The syntax of using new is simple which is given below :

```
data_type *ptr = new data_type;
```

- **<u>The delete operator</u>**

- If we allocate memory when we create an object, we need to de-allocate the memory when the object is no longer needed
- The **delete operator** can be **used to delete memory previously allocated by new operator.**
- The syntax of using delete is simple which is given below :

delete pointer_variable;

- E.g

newdelete.cpp

```
1   #include<iostream>
2   using namespace std;
3   int main()
4   {
5       int *p=new int;
6       *p=20;
7       float *fp=new float(2.34);
8
9       cout<<"int value ="<<*p<<"  Address = "<<p<<endl;
10      cout<<"Float value ="<<*fp<<"  Address = "<<fp<<endl;
11
12      delete p;
13      delete fp;
14      return 0;
15  }
```

- Output:

```
C:\Users\Admin\Desktop\C++Practical\newdelete.exe
int value =20  Address = 0x821540
Float value =2.34  Address = 0x821560


--------------------------------
Process exited after 0.01851 seconds with return value 0
Press any key to continue . . .
```

Malloc Vs. New

- **The new and malloc() both are used to dynamically allocate the memory.**

COMPARISON	NEW	MALLOC ()
Language	The operator new is a specific feature of C++, Java, and C#.	The function malloc () is a feature of C.
Nature	"new" is an operator.	malloc( ) is a function.
Syntax	type reference_variable = **new** type name;	**int** *ptr = (data_type*) malloc(**sizeof**(data_type));
sizeof( )	new doesn't need the sizeof operator as it allots enough memory for specific type.	malloc requires the sizeof operator to know what memory size it has to allot.
Constructor	Operator new can call the constructor of an object.	malloc( ) cannot at all make a call to a constructor.
Initialization	The operator new could initialize an object while allocating memory to it.	Memory initialization could not be done in malloc.
Overloading	Operator new can be overloaded.	The malloc ( ) can never be overloaded.

Failure	On failure, operator new throws an exception.	On failure, malloc ( ) returns a NULL.
Deallocation	The memory allocation by new, deallocated using "delete".	The memory allocation by malloc ( ) is deallocated using a free( ) function.
Reallocation	The new operator does not reallocate memory.	Memory allocated by malloc () can be reallocated using realloc ().
Execution	The operator new cuts the execution time.	The malloc ( ) requires more time for execution.

Pointer Member Operators

- A pointer is a variable that contains the address of the another variable
- Pointer can **point to data types and arrays**
- **A pointers can also point to the member of the class i.e. (The address of a member of a class assign it to a pointer)**

- **The address of the member can be obtained by applying the operator & to a class member name.**
- **A class member pointer can be declared using the operator ::* with the class name.**
- The general syntax is:

data_type class_name :: *ptr_name =&class_name :: member _name;

e.g.

int A:: *ip = &A :: m;

- The **pointer ip** can now be used to access the **member m**. Lets assume that **a** is an **object of class A** .
- Now will see how to access **m using** the **pointer ip**.

```
cout<<a. *ip;   //display m
cout<<a.m;      //same as above
```

- Now ,look at the following code

```
ap = &a;                //ap is pointer to object
cout<<ap -> *ip;   //display m
cout<<ap ->m;   //same as above
```

- The dereferencing operator **.*** is **used** when the **object itself is used with the member pointer**
- The dereferencing operator **->** is **used** to **access a member when we use pointers to both the object and the member.**

E.g.

pointermember.cpp

```cpp
#include <iostream>
using namespace std;
class A
{
    public :
    int m;
};

int main( )
{
    int A :: *ip = &A :: m;
    A a,*ap;

    a.*ip = 30;
    cout<<"\nNumber is (a.m) :"<<a.m;
    cout<<"\nNumber is (a.*ip):"<<a.*ip;

    ap=&a;
    cout<<"\nNumber is (ap->m) :"<<ap->m;
    cout<<"\nNumber is (ab->*ip):"<<ap->*ip;

    return 0;
}
```

- Output:

C:\Users\Admin\Desktop\C++Practical\pointermember.exe

```
Number is (a.m) :30
Number is (a.*ip):30
Number is (ap->m) :30
Number is (ab->*ip):30
--------------------------------
Process exited after 0.021 seconds with return value 0
Press any key to continue . . .
```