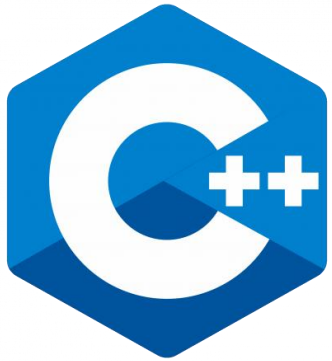


C++ Programming

Trainer : Pradnyaa S. Dindorkar

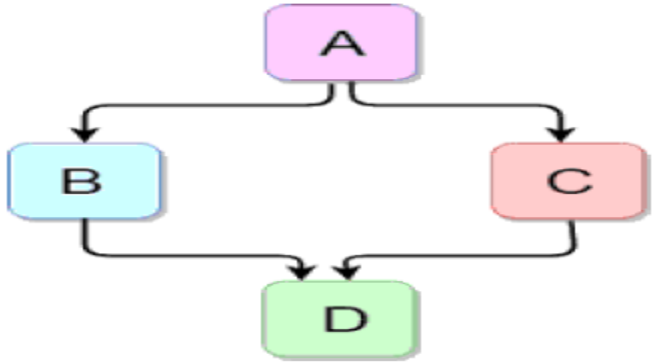
Email: pradnya@sunbeaminfo.com

Today’s topics

1. Diamond problem
2. Virtual function
3. Function overriding
4. Early Binding and late Binding
5. Pure virtual function and Abstract class
6. Template
7. Difference between Procedure Oriented and Object Oriented
8. MCQ

Diamond Problem

- As shown in diagram it is hybrid inheritance. Its shape is like diamond hence it is also called as diamond inheritance.
- Data members of indirect base class inherit into the indirect derived class multiple times. Hence it effects on size of object of indirect derived class.
- Member functions of indirect base class inherit into indirect derived class multiple times. If we try to call member function of indirect base class on object of indirect derived class, then compiler generates ambiguity error.
- If we create object of indirect derived class, then constructor and destructor of indirect base class gets called multiple times.
- All above problems generated by hybrid inheritance is called diamond problem.



Solution to Diamond Problem– Virtual Base Class

- If we want to overcome diamond problem, then we should declare base class virtual i.e. we should derive class B & C from class A virtually. It is called virtual inheritance. In this case, members of class A will be inherited into B & C but it will not be inherited from B & C into class D.

```
class A { };
class B : virtual public A
{ };
class C : virtual public A
{ };
class D : public B, public C
{ };
```

Object slicing

- When a derived class object is assigned to a base class object in C++, the derived class object's extra attributes are sliced off (not considered) to generate the base class object; and this whole process is termed **object slicing**.

- when extra components of a derived class are sliced or not used and the priority is given to the base class's object this is termed object slicing.

- Class base{};
- Class derived :public base {};

```
Main()
 {
        base b ;      derived d ;
        b=d;          //object slicing.
}
```

Virtual Keyword

- **Virtual function** = It is the function which is called depending on type of object rather than type of pointer

- If class contains at least one virtual function then such class is called **polymorphic class**.

- If signature of base class and derived class member function is same and if function in base class is virtual then derived class member function is by default considered as virtual.

Function overriding.

Process of redefining, virtual function of base class, inside derived class with same signature is called function overriding.

- Virtual function redefined inside derived class is called overrided function.
- For function overriding:
    1. Functions must be exist inside base class and derived class.
    2. Signature of base class and derived function must be same.
    3. Function in base class must be virtual.

Program Demo

Early Binding

create a class Base and Derived (void show() in both classes)

create base *bptr;

bptr=&d;

bptr->show()

Late Binding

create a class Base and Derived (void show() in both classes one as virtual in base class)

create base *bptr;

bptr=&d;

bptr->show()

Pure virtual function and Abstract class

- Virtual fun which is equated to zero such function is called as Pure virtual function
- Pure virtual function does not have body.
- A class which contains at least one Pure virtual function such class is called as "Abstract class".
- If class is Abstract we can not create object of that class but we can create pointer or reference of that class .
- It is not compulsory to override virtual function but It is compulsory to override Pure virtual function
- If we not override pure virtual function in derived class at that time derived class can be treated as abstract class.

Upcasting and downcasting

- Upcasting and downcasting :-

Upcasting - process of converting derived class pointer into base class pointer

or Storing address of derived class object into base class pointer.

eg : Person *p=new student();

Downcasting - process of converting base class pointer into derived class pointer

or storing address of base class object into derived class pointer

eg : Student *s=new person();

+ Function Overloading
 - function with same name & different signatures in same scope.
 - No keyword is needed.
 - Call is resolved at compile time depending on args passed.
 - Compile time resolve -- early binding
 - Compile time/static polymorphism
 - Based on "name mangling" feature

+ Function overriding
 - function with same name and same signature redefined in derived class.
 - "virtual" keyword required.
 - Call is resolved at runtime depending on type of object.
 - Runtime resolve -- late binding
 - Run time/dynamic polymorphism
 - Based on "virtual function table" and "virtual function pointer"

Template

- If we want to write generic program in C++, then we should use template.
- This feature is mainly designed for implementing generic data structure and algorithm.
- If we want to write generic program, then we should pass data type as a argument. And to catch that type we should define template.
- Using template we can not reduce code size or execution time but we can reduce developers effort.

int num1 = 10, num2 = 20; swap_object<**int>(num1, num2);** string str1="Pune", str2="Karad"; swap_object<string>(str1, str2);	In this code, <int> and <string> is considered as type argument.

template<typename T> //or
template<class T> //T : Type Parameter
void swap(b obj1, **T** obj2 **)**
{
T temp = obj1;
obj1 = obj2;
obj2 = temp;
}
	template and typename is keyword in C++. By passing datatype as argument we can write generic code hence parameterized type is called template

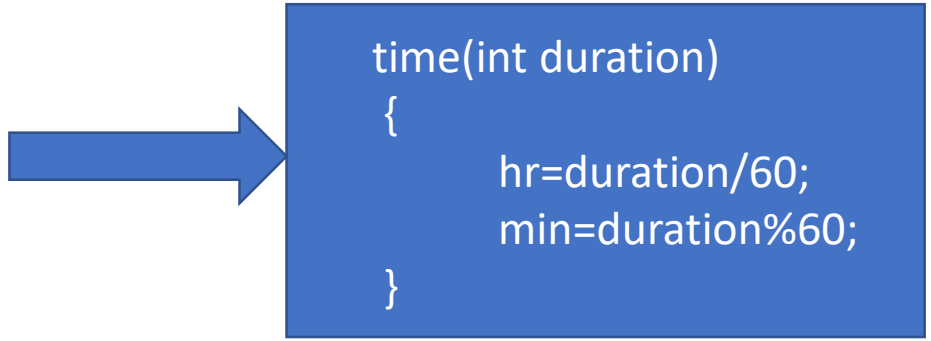
Example of Class Template

```
template<class T>
class Array // Parameterized type
{
private:
    int size;
    T *arr;
    public:
    Array( void ) : size( 0 ), arr( NULL )
    {
    }
    Array( int size )
    {
    this->size = size;
    this->arr = new T[ this->size ];
    }
    void acceptRecord( void ){}
    void printRecord( void ){ }
    ~Array( void ){ }
};
```

```
int main(void)
{
        Array<char> a1( 3 );
        a1.acceptRecord();
        a1.printRecord();
        return 0;
}
```

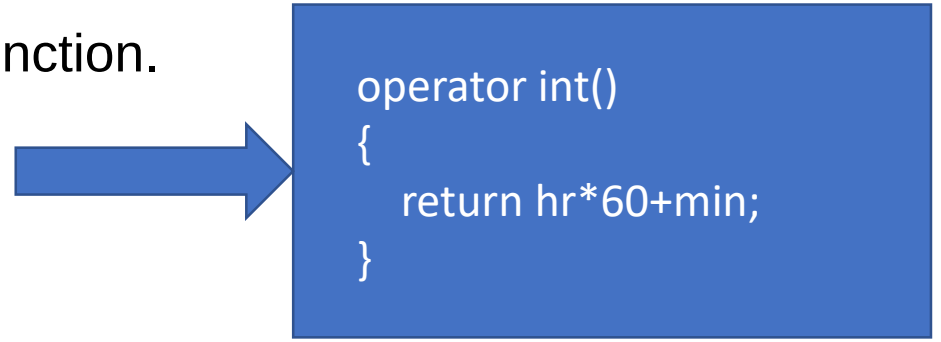
Conversion Function

- You can build the same kind of implicit conversions into your classes by building conversion functions. When you write a function that converts any data type to a class, you tell the compiler to use the conversion function when the syntax of a statement implies that the conversion should take effect, that is, when the compiler expects an object of the class and sees the other data type instead.
- There are two ways to write a conversion function. The first is to write a special constructor function; the second is to write a member conversion function.
- int to object -> constructor act as conversion function



```
time(int duration)
{
        hr=duration/60;
        min=duration%60;
}
```

- Object to int -> operator overloading member conversion function.



```
operator int()
{
   return hr*60+min;
}
```

Smart Pointer

- As we've known unconsciously not deallocating a pointer causes a memory leak that may lead to crash of the program.
- C++ comes up with its own mechanism that's *Smart Pointer* to avoid memory leak.
- When the object is destroyed it frees the memory as well. So, we don't need to delete it as Smart Pointer does will handle it.
- A *Smart Pointer* is a wrapper class over a pointer with an operator like * and -> overloaded.

```
class SmartPtr
{
        rect *ptr;
        public:
        SmartPtr(rect *p = NULL)   {   ptr = p;     }
        ~SmartPtr()     { delete (ptr); }
        rect& operator*()     { return *ptr; }
        rect* operator->()    {   return ptr;  }
};
```

Types of Smart Pointers

1. unique_ptr

   unique_ptr stores one pointer only. We can assign a different object by removing the current object from the pointer.

2. shared_ptr

   By using *shared_ptr* more than one pointer can point to this one object at a time and it'll maintain a Reference Counter using *use_count()* method.

3. weak_ptr

   It's much more similar to shared_ptr except it'll not maintain a Reference Counter. In this case, a pointer will not have a stronghold on the object. The reason is if suppose pointers are holding the object and requesting for other objects then they may form a Deadlock.

Difference between Procedure Oriented and Object Oriented

Procedure Oriented

- Emphasis on steps or algo
- Programs are divided into small code units called Functions.
- Most function shares global data and can modify it
- Data moves from function to function
- Follows Top-down approach
- Example= C

Object Oriented

- Emphasis on data of program
- Programs are divided into small data units called classes
- Data is hidden and not accessible outside the class
- Objects are communicating with each other
- Follows Bottom-up approach
- Example =C++,JAVA,C#.NET, python

Thank You