C++ Programming

Trainer : Rohan Paramane

Email: rohan.paramane@sunbeaminfo.com

Agenda

- Scope
- Macro
- Operator Overloading
- Exception Handling
- Association
  - Composition
  - Aggregation
- Inheritance
- Types of Inheritance

Operator Overloading

- operator is token in C/C++.
- It is used to generate expression.
- operator is keyword in C++.
- 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**

Need Of Operator Overloading

- we extend the meaning of the operator.
- If we want to use operator with the object of use defined type, then we need to overload operator.
- To overload operator, we need to define operator function.
- In C++, operator is a keyword
  - Suppose we want to use plus(+) operator with objects then we need to define operator+( ) function.

We define operator function either inside class (as a member function) or outside class (as a non-member function).	Point pt1(10,20), pt2(30,40), pt3;
	
	pt3 = pt1 + pt2; //pt3 = pt1.operator+(pt2); //using member function
	//or
	pt3 = pt1 + pt2; //pt3 = operator+(pt1, pt2); //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)

Example :

```
Point operator+( Point &other ) //Member Function
    {
    Point temp;
    temp.xPos = this->xPos + other.xPos;
    temp.yPos = this->yPos + other.yPos;
    return temp;
    }
```

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);

Example:

```
Point operator+( Point &pt1, Point &pt2 ) //Non Member Function
{
Point temp;
temp.xPos = pt1.xPos + pt2.xPos;
temp.yPos = pt1.yPos + pt2.yPos;
return temp;
}
```

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(->)

Program Demo without operator overloading

- Create a class point, having two fileld, x, y.
  - Point(void)
  - Point(int x, int y)

```
int main( void )
{
Point pt1(10,20);
Point pt2(30,40);
Point pt3;
pt3 = pt1 + pt2; //Not OK( implicitly)
return 0;
}
```

Exception Handling

- If we give wrong input to the application then it generates runtime error/exception.
- Exception is an object, which is used to send notification to the end user of the system if any exceptional situation occurs in the program.
- To handle exception then we should use 3 keywords:
- 1**. try**
  - try is keyword in C++.
  - If we want to inspect exception then we should put statements inside try block/handler.
  - Try block may have multiple catch block but it must have at least one catch block.

Note : For thrown exception, if we do not provide matching catch block then C++ runtime gives call to the std::terminate() function which implicitly gives call to the std::abort() function.

- **2. catch**
  - If we want to handle exception then we should use catch block/handler.
  - Single try block may have multiple catch block.
  - Catch block can handle exception thrown from try block only.
  - A catch block, which can handle any type of exception is called generic catch block / catch-all handler.
  - For each type of exception, we can write specific catch block or we can write single catch block which can handle all types of exception. A catch block which can handle all type of exception is called generic catch block.
- **3. throw**
  - throw is keyword in C++.
  - If we want to generate exception explicitly then we should use throw keyword.
  - "throw statement" is a jump statement.
  - To generate new exception, we should use throw keyword. Throw statement is jump statement.

- In C++, try, catch and throw keyword is used to handle exception.

```
int num1;
accept_record( num1 );
int num2;
accept_record( num1 );
try
{
if( num2 == 0 )
throw "/ by zero  exception";
int result = num1 / num2;
print_record( result )
}
catch( const char *ex )
{   cout<<ex<<endl;  }
catch(...)
{
cout<<"Genenric catch  handler"<<endl;
}
```

Consider the following code

In this code, int type exception is thrown but matching catch block is not available.

Even generic catch block is also not available. Hence program will terminate.

Because , if we throw exception from try block then catch block can handle it. But with the help of function we can throw exception from outside of the try block.

```
int main( void )
{
int num1;
accept_record(num1);
int num2;
accept_record(num2);
try
{
if( num2 == 0 )
throw 0;
int result = num1 / num2;
print_record(result);
}
catch( const char *ex )
{  cout<<ex<<endl;  }
return 0;
}
```

Consider the following code

If we are throwing exception from function, then implementer of function should specify “exception specification list”. The exception specification list is used to specify type of exception function
may throw.
If type of thrown exception is not available in exception specification list and if exception is raised then C++ do execute catch block rather it invokes std::unexpected() function.

```
int calculate(int num1,int num2) throw(const char* )
{    if( num2 == 0 )
     throw "/ by zero  exception";
     return num1 / num2;
}
int main( void )
{  int num1;
accept_record(num1);
int num2;
accept_record(num2);
try
{    int result = calculate(
num1, num2 );
print_record(result);
}
catch( const char *ex )
{  cout<<ex<<endl; }
return 0; }
```

Scope

- It decides area/region/boundry in which we can access the element.
- **Types of scope in C++:**
  1. **Block scope**
  2. **Function scope**
  3. **Prototype scope**
  4. **Class scope**
  5. **Namespace scope**
  6. **File scope**
  7. **Program scope**

Example Scope

```
int num6;           //Program Scope
static int num5;    //File Scope
namespace ntest
{  int num4;    //Namespace scope
   class Test
   {   int num3;    //Class Scope  };
}
void sum( int num1, int num2 ); //Prototype scope
int main( void )
{
   int num1 = 10; //Function Scope
   while( true )
   {  int temp = 0; }
   return 0;    //Block Scope
}
```

Macro

- Symbolic constant is called as macro
- Expanding macro is a job of pre processor.
- Example:
  - #define SIZE 10
  - #define EOF -1
  - **#define MULTIPLY(x,y) x*y**
- **Few other Macro's in C++**
  - __FILE__, __LINE__,__DATE__,__TIME__

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:

```
class Engine
{ };
class Car
{               private:
    Engine e;   //Association
};
int main( void )
{   Car car;
    return 0;
}
//Dependant Object : Car Object
//Dependancy Object : Engine Object
```

Composition and aggregation are specialized form of association

Composition

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

```
class Heart
{ };
class Human
{  Heart hrt; //Association->Composition
};
int main( void )
{  Human h;
   return 0;
}
```

- //Dependant Object : Human Object
- //Dependancy Object : Heart Object

Aggregation

- If dependency object exist without Dependant object then it represents Aggregation.
- Aggregation represents loose coupling.

```
class Faculty
{ };
class Department
{
   Faculty f; //Association->Aggregation
};
int main( void )
{
   Department d;
   return 0;
}
```

- //Dependant Object : Department Object
- //Dependancy Object : Faculty Object

Inheritance

- If "is-a" relationship exist between two types then we should use inheritance.
- Inheritance is also called as "Generalization".
- Example: Book is-a product
- During inheritance, members of base class inherit into derived class.
- If we create object of derived class then non static data members declared in base class get space inside it.
- Size of object = sum of size of non static data members declared in base class and derived class.
- If we use private/protected/public keyword to control visibility of members of class then it is called access Specifier.
- If we use private/protected/public keyword to extend the class then it is called mode of inheritance.
- Default mode of inheritance is private.
  - Example: class Employee : person //is treated as class Employee : private Person
- Example: class Employee:public Person
- In all types of mode, private members inherit into derived class but we can not access it inside member function of derived class.
- If we want to access private members inside derived class then:
  - Either we should use member function(getter/setter).
  - or we should declare derived class as a friend inside base class.

Syntax of inheritance in C++

class Person //Parent class { }; **class Employee : public Person //** Child class { };	In C++ Parent class is called as Base class and child class is called as derived class. To create derived class we should use colon(:) operator. As shown in this code, public is mode of inheritance.

class Person //Parent class
{ **char name[30]; int age;** };
class Employee : public Person //Child class
{ **int empid; float salary;** };
int main(void)
{
Person p;
cout<<**sizeof(p)<<endl;**
Employee emp;
cout<<**sizeof(emp)<<endl;**
return 0;
}
	If we create object of derived class, then all the non- static data member declared in base class & derived class get space inside it i.e. non-static static data members of base class inherit into the derived class.

Syntax of inheritance in C++

- Using derived class name, we can access static data member declared in base class i.e. static data member of base class inherit into derived class.

```
class Base{
protected:
static int number;
};
int Base::number = 10;
class Derived : public Base{
public:
static void print( void )
{ cout<<Base::number<<endl; }
};
int main( void ){
Derived::print();
return 0;
}
```

```
class Derived : public Base
{
int num3;
static int num4;
public:
void setNum3( int num3 )
{  this->num3 = num3; }
static void setNum4( int num4 )
{  Derived::num4 = num4; }
};
int Derived::num4;
```

```
int main( void )
{
Derived d;
d.setNum1(10);
d.setNum3(30);
Derived::setNum2(20);
Derived::setNum4(40);
return 0;
}
```

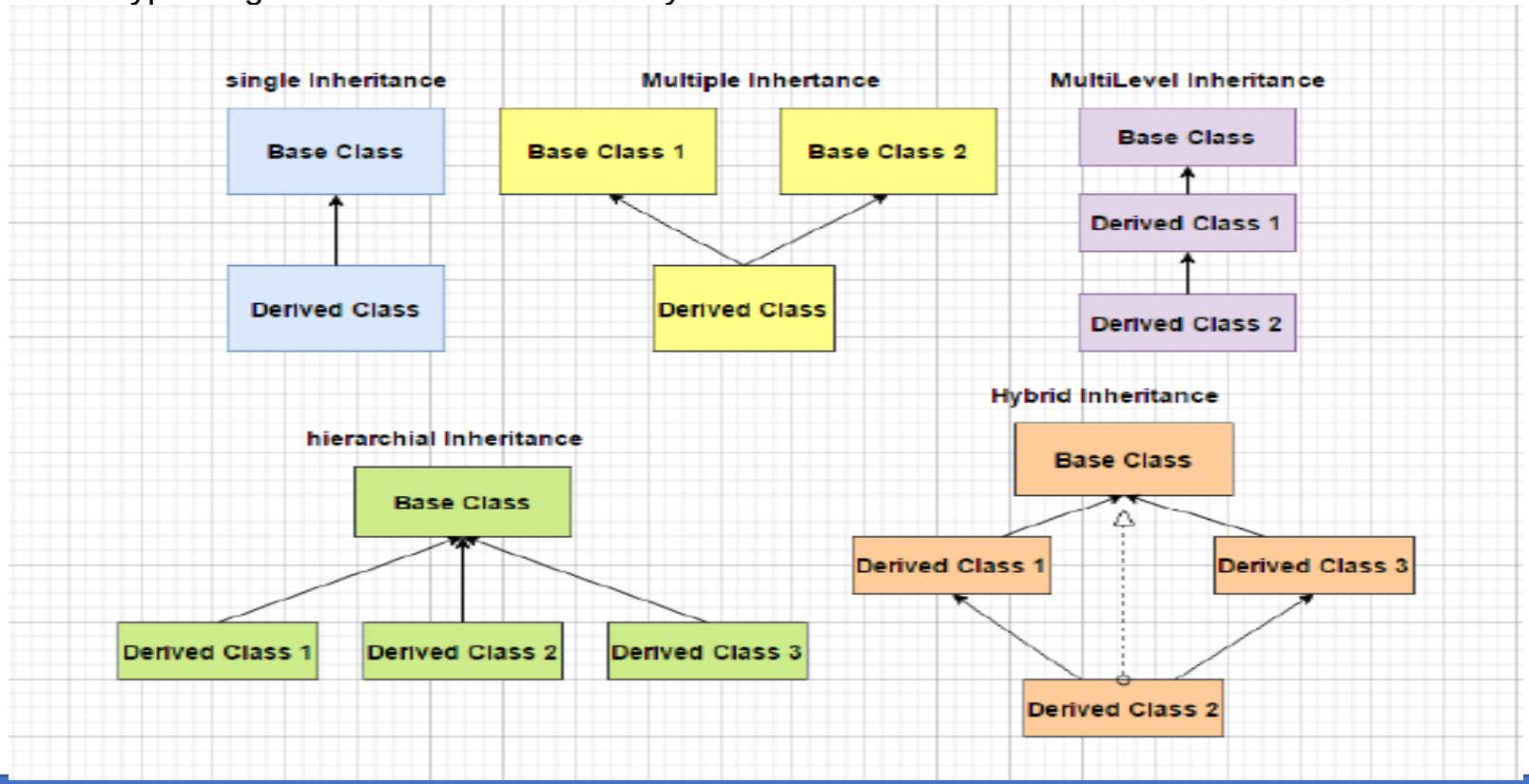
Except following functions, including nested class, all the members of base class, inherit into the derived class

- Constructor
- Destructor
- Copy constructor
- Assignment operator
- Friend function.

Types of Inheritance

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance

If we combine any two or more types together then it is called as hybrid inheritance.



Thank You