11 References & the Copy-Constructor

Hu Sikang
skhu@163.com

Contents

- Reference
- Copy Constructor

11.1 References in C++

- A *reference* (**&**) is usually used for function **argument lists** and function return values.
- When a reference is used as a **function argument**, any modification to the reference *inside* the function will cause changes to the argument *outside* the function.
- If you return a reference from a function, you must take the same care as if you return a pointer from a function, avoiding to refer to unknown memory.

11.1.1 References in functions

```
// Simple C++ references
int* f(int* x)
{
    (*x)++;
    // Safe, x is outside
    return x;
}

int& g(int& x)
{
    x++;          // Same effect as in f()
    return x; // Safe
}
```

```
void main()
{
    int a = 0, b;
    int *p;
    // Ugly (but explicit)
    p = f(&a);

    // Clean (but hidden)
    b = g(a);
}
```

11.1.2 References with local varaible

```
int& h( )
{
   int q = 10;
   return q;  // Error
 }

void main( )
{
   int &a = h( );
   int &b = h( );
   b++;
   cout << a << endl;
 }
```

```
int* h( )
{
        int q  = 20;
        return &q;  // Error
 }

void main( )
{
        int* p;
        p = h( );
        cout << *p << endl;
 }
```

11.2 The copy-constructor

Initialize the object with other object of the same class.

int x = 10;

int y(x);

class Sample;

Sample S1;

*Sample **S2(S1)**;*

*Sample **S2 = S1**;*

11.2 The copy constructor

There is a copy-constructor in a class. The genetic form of copy-constructor is:

```
class Sample
{
public:
    Sample(const Sample&);
};
```

The executing semantic of copy-constructor is to initialize object member data.

11.2.1 Copy Constructors

（1）initialize object

```
class  Point {
 public :
   Point ( int  xx = 0 ,  int  yy = 0 )
   { X = xx ;  Y = yy ; }
   Point ( const Point  & p ) ;
   int  GetX ( ) { return  X ; }
   int  GetY ( ) { return  Y ; }
 private :   int  X ,  Y ;
};
Point :: Point ( const Point& p)
{      X=p.X;  Y=p.Y;
 cout << "Copy-constructor called." << endl ;
}
```

```
void main ( ) {

  Point  A ( 1 , 2 ) ;
  //call copy-constructor
  Point  B ( A ) ;

  cout << "B:" << B.GetX ( );

  cout  << B.GetY ( ) << endl ;

}
```

11.2.1 Copy Constructors

（1）initialize object

The default one is created by the compiler automatically when a copy- constructor don't be defined by user.

```
class  Point {
 public :
   Point ( int  xx = 0 ,  int  yy = 0 ) { X = xx ;  Y = yy ; }
   int  GetX ( ) { return  X ; }
   int  GetY ( ) { return  Y ; }
 private :   int  X ,  Y ;
};

void main ( ) {
 Point A ( 1 , 2 ) ;
 Point  B ( A ) ;   //call default copy constructor
 cout << "B:" << B.GetX ( ) << "," << B.GetY ( ) << endl ;
}
```

11.2.1 Copy Constructors

（2）When the argument is a object, copy-constructor is called.

```
class  Point {
 public :
   Point ( int  xx = 0 ,  int  yy = 0 ) { X = xx ;  Y = yy ; }
   Point ( const Point & p ) ;
   ~ Point ( ) { cout << X << "," << Y << " Object destroyed." << endl ; }
   int  GetX ( ) { return  X ; }              int  GetY ( ) { return  Y ; }
 private :   int  X ,  Y ;
} ;
Point :: Point ( const Point & p)                    // call copy constructor
{ X = p.X ;  Y = p.Y ;  cout << "Copy-constructor called." << endl ; }
void  f ( Point  p ) { cout << "Funtion:" << p.GetX() <<"," << p.GetY() << endl ; }
void main ( )
 {   Point  A ( 1 , 2 ) ;          f ( A ) ; }
```

11.2.1 Copy Constructors

（3）When the returning type of function is class type, copy constructor is called.

```
class  Point {
 public :
   Point ( int  xx=0,  int yy=0 )
      { X=xx ; Y=yy;  cout << "Object constructed." << endl ; }
   Point ( const Point & p ) ;
   ~ Point ( ) { cout << X << "," << Y << " Object destroyed." << endl ; }
   int  GetX ( ) { return  X ; }              int  GetY ( ) { return  Y ; }
 private :   int   X , Y ;
} ;
Point :: Point ( const Point & p )
{  X= p.X ;  Y=p.Y ;  cout << "Copy_constructor called." << endl ; }
Point  g ( ) { Point  A ( 1 , 2 ) ;  return  A ; }
void main ( ) {  Point  B ;    B = g ( ) ; }
```

output:　　remark:

Implementation Analysis

```
class  Point {
 public :
   Point ( int  xx=0,  int yy=0 )
      { X=xx ; Y=yy;  cout << "Object constructed. " << endl ; }
   Point ( const Point & p ) ;
   ~Point( ) { cout << X << ", " << Y << " Object destroyed. " << endl ; }
   int  GetX ( ) { return  X ; }          int  GetY ( ) { return  Y ; }
 private :   int   X , Y ;
} ;
Point :: Point ( const Point & p )
{  X= p.X ;  Y=p.Y ;  cout << "Copy_constructor called. " << endl ; }

Point  g ( ) { Point  A ( 1 , 2 ) ;  return  A ; }

void main ( ) {  Point  B ;    B = g ( ) ; }
```

output:

```
Object constructed.
```

remark:

Create a B's object

Implementation Analysis

```
class  Point {
 public :
   Point ( int  xx=0,  int yy=0 )
      { X=xx ; Y=yy;  cout << "Object constructed." << endl ; }
   Point ( const Point & p ) ;
   ~Point( ) { cout << X << "," << Y << " Object destroyed." << endl ; }
   int  GetX ( ) { return  X ; }              int  GetY ( ) { return  Y ; }
 private :   int   X , Y ;
} ;
Point :: Point ( const Point & p )
{  X= p.X ;  Y=p.Y ;  cout << "Copy_constructor called." << endl ; }

Point  g ( ) { Point  A ( 1 , 2 ) ;  return  A ; }

void main ( ) { Point  B ;    B = g ( ) ; }
```

output:

```
Object constructed.
Object constructed.
```

remark:

Create a B's object

Create a local A's object

Implementation Analysis

```
class  Point {
 public :
   Point ( int  xx=0,  int yy=0 )
      { X=xx ; Y=yy;  cout << "Object constructed." << endl ; }
   Point ( const Point & p ) ;
   ~Point( ) { cout << X << "," << Y << " Object destroyed." << endl ; }
   int  GetX ( ) { return  X ; }              int  GetY ( ) { return  Y ; }
 private :   int   X , Y ;
} ;
Point :: Point ( const Point & p )
{  X= p.X ;  Y=p.Y ;  cout << "Copy_constructor called." << endl ; }

Point  g ( ) { Point  A ( 1 , 2 ) ;  return  A ; }

void main ( ) { Point  B ;    B = g ( ) ; }
```

output:

```
Object constructed.
Object constructed.
Copy_constructor call.
```

remark:

Create a B's object

Create a local A's object

Return an anonymous object

Implementation Analysis

```
class  Point {
 public :
   Point ( int  xx=0,  int yy=0 )
      { X=xx ; Y=yy;  cout << "Object constructed." << endl ; }
   Point ( const Point & p ) ;
   ~Point( ) { cout << X << "," << Y << " Object destroyed." << endl ; }
   int  GetX ( ) { return  X ; }          int  GetY ( ) { return  Y ; }
 private :   int   X , Y ;
} ;
Point :: Point ( const Point & p )
{  X= p.X ;  Y=p.Y ;  cout << "Copy_constructor called." << endl ; }

Point  g ( ) { Point  A ( 1 , 2 ) ;  return  A ; }

void main ( ) { Point  B ;    B = g ( ) ; }
```

output:

```
Object constructed.
Object constructed.
Copy_constructor call.
```

remark:

Create a B's object

Create a local A's object

Return an anonymous object

Implementation Analysis

```
class  Point {
 public :
   Point ( int  xx=0,  int yy=0 )
      { X=xx ; Y=yy;  cout << "Object constructed." << endl ; }
   Point ( const Point & p ) ;
   ~Point( ) { cout << X << "," << Y << " Object destroyed." << endl ; }
   int  GetX ( ) { return  X ; }          int  GetY ( ) { return  Y ; }
 private :   int   X , Y ;
} ;
Point :: Point ( const Point & p )
{  X= p.X ;  Y=p.Y ;  cout << "Copy_constructor called." << endl ; }

Point  g ( ) { Point  A ( 1 , 2 ) ;  return  A ; }

void main ( ) { Point  B ;    B = g ( ) ; }
```

Evluate to object B

output:

```
Object constructed.
Object constructed.
Copy_constructor call.
1 , 2 Object destroyed.
1 , 2 Object destroyed.
1 , 2 Object destroyed.
```

remark:

Create a B's object

Create a local A's object

Return an anonymous object

Release local object A

Release anonymous object

Release object B

Implementation Analysis

```
class Point {
 public :
   Point ( int xx=0
      { X=xx ; Y=y                                  << (
   Point ( const Po
   ~Point( ) { cout << X << "," << Y << " Object destroyed." << endl ; }
   int GetX ( ) { return X ; }          int GetY ( ) { return Y ; }
 private : int X , Y ;
} ;
Point :: Point ( const Point & p )
{ X= p.X ; Y=p.Y ; cout << "Copy_constructor called." << endl ; }

Point g ( ) { Point A ( 1 , 2 ) ; return A ; }

void main ( ) { Point B ;   B = g ( ) ; }
```

Evaluate to object B

11.2.1 Copy Constructors

```
class  Point {
 public :
   Point ( int  xx=0,  int yy=0 )
      { X=xx ; Y=yy;  cout << "Object constructed." << endl ; }
   Point ( const Point & p ) ;
   ~Point( ) { cout << X << "," << Y << " Object destroyed." << endl ; }
   int  GetX ( ) { return  X ; }              int  GetY ( ) { return  Y ; }
 private :   int   X , Y ;
} ;
Point :: Point ( const Point & p )
{  X= p.X ;  Y=p.Y ;  cout << "Copy_constructor called." << endl ; }

Point  g ( ) { Point  A ( 1 , 2 ) ;  return  A ; }

void main ( ) { Point B ;    B = g ( ) ; }
```

```
void main ( ) {   Point B = g ( ) ;   }
```

11.2.2 Exercise

```
// tpoint.h
class TPoint
{
public:
        TPoint(int x,int y) {X=x;Y=y;}
        TPoint(TPoint& p);                  //copy constructor
        ~TPoint() {cout<<"Destructor is called."<<endl;}
        int Xcoord() {return X;}
        int Ycoord() {return Y;}
private:
        int X,Y;
};
TPoint::TPoint(TPoint& p)
{
        X=p.X;
        Y=p.Y;
        cout<<"Copy Constructor is called.\n";
}
```

11.2.2 Exercise

```
// tpoint.cpp
TPoint fun(TPoint Q)    // Pass by data value
 {     cout<<“In fun()! "<<endl;
       int x, y;
       x=Q.Xcoord()+10;
       y=Q.Ycoord()+20;
       TPoint R(x,y);
       return R;        // Return R’s data value
 }
 void main()
 {      TPoint M(20,35),P(0,0);
       TPoint N(M);    //M is a created object, N is a creating object
       P=fun(N);
       cout<<"P="<<P.Xcoord()<<", "<<P.Ycoord()<<endl;
 }
```

11.2.2 Exercise

Output:

```
Copy  Constructor is called.
Copy  Constructor is called.
In fun()!
Copy  Constructor is called.
Destructor is called.
Destructor is called.
Destructor is called.      // temporary object
P=30,55
Destructor is called.
Destructor is called.
Destructor is called.
```

11.2.3 Copy Constructors

- The Copy Constructor is called when **passing arguments by object**.
- The Copy Constructor is NOT called when **passing arguments by references or by pointers**. Because no new object is created.
- The Copy Constructor is called when function **returning an object**.