Lecture Five

Function Overloading

Ref: Herbert Schildt, Teach Yourself C++, Third Edn (Chapter 5)

© Dr. M. Mahfuzul Islam
Professor, Dept. of CSE, BUET

Overloading Constructor Functions

- It is common to overload a class's constructor function.
- It is not possible to overload a destructor function.
- Three main reasons to overload constructor function:
  - to gain flexibility,
  - to support arrays and
  - to create copy constructors.
- If a program attempts to create an object for which no matching constructor is found, a compile-time error occurs.

Overloading Constructor Functions

Example for gaining flexibility

```
#include <iostream>
#include <cstdio>
using namespace std;

class date {
      int month, day, year;
public:
      date(char *str);
      date( int d, int m, int y) ){
            day = d;
            month = m;
            year = y;
      }
      void show(){
            cout << day << '/' << month << '/' ;
            cout << year << '\n';
      }
};
```

```
date::date(char *str){
      sscanf(str, "%d%*c%d%*c%d", &day,
                              &month, &year);
}

int main() {
      date sdate("31/12/99");
      date idate(31, 12, 99);

      sdate.show();
      idate.show();
      return 0;
}
```

Overloading Constructor Functions

Example for supporting array

```
#include <iostream>
using namespace std;

class myclass {
      int x;
public:
      myclass() { x = 0;}
      myclass(int n) { x = n;}
      int getx() {return x;}
};
```

```
int main() {
     myclass o1[10];
     myclass o2[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9,
                                          10};

     for(int i = 0; i < 10; i++){
          cout << "o1[" << i << "]: " << o1[i].getx();
          cout << '\n';
          cout << "o2[" << i << "]: " << o2[i].getx();
          cout << '\n';
     }
     return 0;
}
```

Creating and Using copy constructor

- **Problems can occur when an object is passed to or returned from a function. "copy constructor" is one of the solutions.**
- **There are two distinct situations for assigning one object to another- assignment and initialization. The copy constructor only applies to initializations. It does not apply to assignments.**
- **Common form of assignment:**

```
classname (const classname &object){}
```

Creating and Using copy constructor

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;

class strtype{
      char *p;
public:
      strtype(char *s);
      ~strtype() { delete [] p;}
      char *get() { return p; }
};

strtype:: strtype(char *s){
       int l;

       l = strlen(s) + 1;
       p = new char[l];
       if (!p){
             cout << "Allocation error\n";
             exit(1);
        }
       strcopy( p, s );
}
```

```
void show(strtype x){
      char *s;

       s = x.get();
       cout << s << '\n';
}

int main(){
      strtype a("Hello"), b("There");

       show(a);
       show(b);
       return 0;
}
```

What is the problem of the program?

Creating and Using copy constructor

Solving the problem using "COPY CONSTRUCTOR"

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;

class strtype{
      char *p;
public:
      strtype(char *s);
      strtype(const strtype &o);
      ~strtype() { delete [] p;}
      char *get() { return p; }
};

strtype:: strtype(char *s){
       int l;

       l = strlen(s) + 1;
       p = new char[l];
       if (!p){
             cout << "Allocation error\n";
             exit(1);
        }
       strcopy( p, s );
}

strtype:: strtype(const strtype &o){
       int l;
       l = strlen(o.p) + 1;
       p = new char[l];
       if (!p){
             cout << "Allocation error\n";
             exit(1);
       }
       strcopy( p, o.p );
}

void show(strtype x){
       char *s;

        s = x.get();
        cout << s << '\n';
}

int main(){
       strtype a("Hello"), b("There");

        show(a);
        show(b);
        return 0;
}
```

Creating and Using copy constructor

➢**The copy constructor is invoked when a function generates the temporary object.**

```
#include <iostream>
using namespace std;
```

```
class myclass {
public:
        myclass();
        myclass(const myclass &o);
        myclass f();
};
```

```
myclass:: myclass(){
      cout << "Constructing normally\n";
}

myclass:: myclass(const myclass &o){
      cout << "Constructing copy\n";
}
```

```
myclass myclass::f(){
       myclass temp;
       return temp;
}
```

```
int main(){
      myclass obj;

      obj = obj.f();
      return 0;
}
```

OUTPUT:

Constructing normally
Constructing normally
Constructing copy

The Overload Anachronism

- **When C++ was first invented, the keyword overload was used to create an overloaded function.**
- **Overload is obsolete now and no longer supported by modern C++ compilers.**

The general form of overload

overload func-name;

Overloading a function called timer():

overload timer;

Using Default Arguments

- **The defaults can be specified either in function prototype or in its definition if the definition precedes the function's first use.**
- **The defaults cannot be specified in both the prototype and the definition.**
- **All default parameters must be to the right of any parameters that do not have defaults.**
- **Default arguments must be constants or global variables. They cannot be local variables or other parameters.**

```
#include <iostream>
using namespace std;

void f(int a = 0, int b = 0){
      cout << a << “  “ << b << ‘\n\;
}

int main(){
      f();
      f(10);
      f(10, 99);
}
```

Output:

```
0   0
10   0
10  99
```

```
void f(int a = 0, int b){
      cout << a << “  “ << b <<’\n’;
}
```

Wrong! b must have default, too

Using Default Arguments

- **Default argument can be used instead of function overload**

```
#include <iostream>
using namespace std;

double rect_area( double length, double width = 0){
        if (!width) width = length;
        return length*width;
}

int main(){
        cout << rect_area(10.0, 5.8) << '\n';
        cout << rect_area(10.0) <<'\n';
        return 0;
}
```

```
#include <iostream>
using namespace std;

class myclass {
        int x;
public:
        myclass(int n = 0) { x = n; }
        int getx() { return x; }
};

int main(){
        myclass o1(10);
        myclass o2;

        cout << o1.getx() << '\n';
        cout << o2.getx() << '\n';
        return 0;
}
```

- **It is possible to create copy constructors that take additional arguments, as long as the additional arguments have default values.**

```
myclass( const myclass &obj, int x = 0){
        //body of constructor
}
```

Overloading and Ambiguity

➢**Automatic type conversion rule cause an ambiguous situation.**

```
#include <iostream>
using namespace std;

float f(float i){
      return i / 2.0;
}

double f(double i){
      return i / 3.0;
}
```

```
int main(){
      float x = 10.09;
      double y = 10.09;

       cout << f(x)<<'\n';
       cout << f(y)<<'\n';
       cout << f(10)<<'\n';     // ambiguous
       return 0;
}
```

➢**Wrong type of arguments causes an ambiguous situation.**

```
#include <iostream>
using namespace std;

void f(unsigned char c){
       cout << c;
}

void f(char c){
       cout << c;
}
```

```
int main(){
       f('c');
       f(86);   // which f() is called?
       return 0;
}
```

Overloading and Ambiguity

- **Call by value and call by reference cause an ambiguous situation.**

```
#include <iostream>
using namespace std;

int f(int a, int b){
    return a+b;
}

int f(int a, int &b){
    return a-b;
}
```

```
int main(){
    int x = 1, y = 2;

    cout << f(x, y);  // which f() is called?
    return 0;
}
```

- **Default argument causes an ambiguous situation.**

```
#include <iostream>
using namespace std;

int f(int a){
    return a*a;
}

int f(int a, int b = 0){
    return a*b;
}
```

```
int main(){
    cout << f(10, 2);
    cout << f(10);  // which f() is called?
    return 0;
}
```

Finding address of an Overloaded Function

- **A function address is obtained by putting its name on the right side of an assignment statement without any parenthesis or arguments.**

  To assign p the address of zap(),

  p = zap;

- **What about overloaded function????**

```
#include <iostream>
using namespace std;

void space(int count){
      for( ; count; count--) cout << ' ';
}

void space(int count, char ch){
      for( ; count; count--) cout << ch;
}
```

```
int main(){
      void (*fp1)(int);
      void (*fp2)(int, char);

       fp1 = space;
       fp2 = space;

       fp1(22);
       cout << '\n';

       fp2(30, 'x');
       cout << '\n';

       return 0;
}
```