C++ Function Overloading (Polymorphism)

Dr. Md. Humayun Kabir
CSE Department, BUET

Function Overloading

```
#include <iostream>
using namespace std;
int average_i(int [], int);
double average_d(double [], int);
int main(){
    int members[]={23, 31, 53, 41,93};
    double ranks[]={0.3, 0.1, 0.3, 0.2, 0.5};
    cout<<“Average members: “<<average_i(members, 5)<<endl;
    cout<<“Average rank: “<<avergae_d(ranks, 5)<<endl;
    return 0;
}
```

Function Overloading

```
int average_i(int arr[], int size){
	int i, avg=0;
	for(i=0;i<size;i++){
		avg+=arr[i];
	}
	return avg/size;
}
double average_d(double arr[],int size){
	int i;
	double avg=0;
	for(i=0;i<size;i++){
		avg+=arr[i];
	}
	return avg/size;
}
```

Function Overloading

```
#include <iostream>
using namespace std;
int average(int [], int);
double average(double [], int);
int main(){
    int members[]={23, 31, 53, 41,93};
    double ranks[]={0.3, 0.1, 0.3, 0.2, 0.5};
    cout<<“Average members: “<<average(members, 5)<<endl;
    cout<<“Average rank: “<<average(ranks, 5)<<endl;
    return 0;
}
```

Function Overloading

```
int average(int arr[], int size){
	int i, avg=0;
	for(i=0;i<size;i++){
		avg+=arr[i];
	}
	return avg/size;
}
double average(double arr[],int size){
	int i;
	double avg=0;
	for(i=0;i<size;i++){
		avg+=arr[i];
	}
	return avg/size;
}
```

Function Overloading

- Two or more overloaded functions can have the same name
- However, either the argument types or the number of the arguments or both of the overloaded functions must differ
- Return type of the overloaded functions might be the same or different
- Return type alone is not a sufficient difference to allow function overloading
- Reduces the complexity of a program by allowing related operations to be referred to by the same name

Function Overloading

```
#include <iostream>
using namespace std;
class Math{
    public:
    int square(int);
    double square(double);
};
int Math::square(int n){
    return n*n;
}
double Math::square(double d){
    return d*d;
}
```

Function Overloading

```
int main(){
    int i=12;
    double d=3.9;
    Math m;
    cout<<"The square of "<<i<<" is "<<m.square(i)<<endl;
    cout<<"The square of "<<d<<" is "<<m.square(d)<<endl;
    return 0;
}
```

Constructor Function Overloading

- Constructor function can be overloaded for three reasons
  - To gain flexibility
  - To support array
  - To create copy constructor
- It is not possible to overload the destructor function

Constructor Function Overloading

```
#include <iostream>
using namespace std;
class CRectangle {
    int width, height;
    public:
        CRectangle () {width=0, height=0;}
        CRectangle (int,int);
        void set_values (int,int);
        int area () {return (width*height);}
};
CRectangle::CRectangle (int a, int b) {
    width = a; height = b;
}
void CRectangle::set_values (int a,int b){
    width = a; height = b;
}
```

Constructor Function Overloading

```
int main () {
    CRectangle recta;
    CRectangle rectb (5,6);
    CRectangle rects[2];
    //CRectangle rects[2]={CRectangle(7,8), CRectangle(12,4)};
    recta.set_values(3,4);
    rects[0].set_values(7,8);
    rects[1].set_values(12,4);
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    cout << "rects[0] area: " << rects[0].area() << endl;
    cout << "rects[1] area: " << rects[1].area() << endl;
    return 0;
}
```

Constructor Function Overloading

```
int main () {
    CRectangle *rects=new CRectangle[2] ;
    rects->set_values(7,8);
    cout << "rects[0] area: " << rects->area() << endl;
    rects++;
    rects->set_values(12,4);
    cout << "rects[1] area: " << rects->area() << endl;
    return 0;
}
```

Copy Constructor

- **Copy constructor** is a constructor which creates an object by initializing it with an object of the same class
- The copy constructor is used to:
  - Initialize one object from another object of the same type in a declaration statement
  - Copy an object to pass it as an argument to a function
  - Copy an object to return it from a function

Copy Constructor

- If a copy constructor is not defined in a class, the compiler itself defines one (shallow or default copy constructor), which makes a shallow copy of the object.
- If the class has pointer variables and dynamic memory allocations, then shallow or bitwise copy is not enough
- It must make a deep copy with an explicitly defined copy constructor

Copy Constructor

```
#include <iostream>
using namespace std;
class CRectangle {
    int *width, *height;
    public:
        CRectangle ();
        CRectangle (int,int);
        ~CRectangle ();
        int area () {return (*width * *height);}
};
CRectangle::CRectangle () {
    width = new int;
    height = new int;
    *width = 0;
    *height = 0;
}
```

Copy Constructor

```
CRectangle::CRectangle (int a, int b) {
     width = new int;
    height = new int;
    *width = a;
    *height = b;
}

CRectangle::~CRectangle () {
    delete width;
    delete height;
}
```

Copy Constructor

```
CRectangle larger(CRectangle ra, CRectangle rb){
    if(ra.area()>rb.area())
        return ra;
    else
        return rb;
}
```

Copy Constructor

```
int main () {
    CRectangle recta (3,4), rectb(5,6);
    CRectangle rectc=recta; //this will cause problem
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    cout << "rectc area: " << rectc.area() << endl;
    /*calling larger(recta rectb) will cause problem */
    cout<< “large area: “<<larger(recta,rectb).area();
    return 0;
}
```

Copy Constructor

```
class CRectangle {
    int *width, *height;
    public:
        CRectangle ();
        CRectangle (int,int);
        CRectangle(const CRectangle &r);
        ~CRectangle ();
        int area () {return (*width * *height);}
};
```

Copy Constructor

```
CRectangle::CRectangle(const CRectangle &r){
    width=new int;
    height=new int;
    *width=*r.width;
    *height=*r.height;
}
```

Copy Constructor

```
int main () {
    CRectangle recta (3,4), rectb(5,6);
    CRectangle rectc=recta; //this will call copy constructor
    //CRectangle rect_larger;
    //CRectangle rectd;
    //rectd=rectb;
    /*Following statement will call copy constructor and cause
    the program to crash
    */
    //rect_larger=larger(recta, rectb);
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    cout << "rectc area: " << rectc.area() << endl;
    /*this will call both copy constructor and destructor 3 times */
    cout<< “large area: “<<larger(recta,rectb).area();
    //cout << "rect_larger area: " << rect_larger.area() << endl;
    return 0;
}
```

cout<<“Thank You”<<endl;

cout<<“**Have a Good Day**”<<endl;