



## 7. Function Overloading & Default Arguments

Hu Sikang  
*skhu@163.com*

School of Computer  
Beijing Institute of Technology



# Contents

- Name decoration
- Default arguments
- function overloading



## 7.1 Name Decoration

- ◆ When some functions perform *the same task on objects of different types*, it can be more convenient to give them the same name.
- ◆ Using the *same name* for operations on different types is called *overloading*.
- ◆ Invoke the functions that is the *best match* on the arguments:
  - ◆ the *number* of the arguments;
  - ◆ the *type* of the arguments;



// The number of parameters are different

*int min(int a, int b)*

```
{  
    return a < b ? a : b;  
}
```

*int min(int a, int b, int c)*

```
{  
    int t = min(a, b);  
    return min(t, c);  
}
```

void main()

```
{  
    cout << min(-2, 8, 0) << endl;  
    cout << min(0,8) << endl;  
}
```



//The type of arguments are different

*int min(int a, int b)*

```
{  
    return a < b ? a : b;  
}
```

*double min(double a, double b)*

```
{  
    return a < b ? a : b;  
}
```

void main()

```
{  
    cout << min(23, 87) << endl;  
    cout << min(0.538, 8.72) << endl;  
}
```



```
// Function overloading
```

```
#ifndef STASH3_H
```

```
#define STASH3_H
```

```
class Stash {
```

```
    int size;           // Size of each space
```

```
    int quantity;       // Number of storage spaces
```

```
    int next;           // Next empty space
```

```
    unsigned char* storage; // Dynamically allocated array of bytes
```

```
    void inflate(int increase);
```

```
public:
```

```
    Stash(int size); // Zero quantity
```

```
    Stash(int size, int initQuantity);
```

```
    ~Stash();
```

```
    int add(void* element);
```

```
    void* fetch(int index);
```


```
    int count();
```

```
};
```

```
#endif // STASH3_H
```



## 7.2 Default arguments

- ◆ **Parameters can be assigned default values.**
  - ◆ **Parameters assume their default values when no actual parameters are specified for them in a function call.**
- 



## 7.2 Default arguments

Here is a definition of *min* function.

```
int min(int a = 10, int b=20)
{
    return a < b ? a : b;
}
```

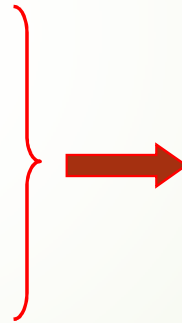
// So it can be called as follows:

```
void main()
{
    int x = min(22, 33);
    int y = min(22);      // Equals to min(22, 20)
    int z = min();        // Equals to min(10, 20)
}
```



## 7.2 Default arguments

```
class Point
{
public:
    Point( )
    { coordX = 0; coordY = 0; }
    Point(double x, double y)
    { coordX = x; coordY = y; }
private:
    double coordX, coordY;
};
```



```
class Point
{
public:
    Point(double x = 0,
           double y = 0)
    { coordX = x;
      coordY = y; }
private:
    double coordX,
           coordY;
};
```



## 7.2 Default Arguments

// Find the sum of numbers in a range of values

// Between “lower” and “upper” using increment “inc”

```
#include <iostream>
```

```
using namespace std;
```

```
int sum(int lower, int upper=10, int inc=1) {
```

```
    int sum = 0;
```

```
    for(int k = lower; k <= upper; k += inc)        sum += k;
```

```
    return sum;
```

```
}
```

```
void main() {
```

```
    int result = sum(1);
```

```
    cout << "The sum from 1 to 10 step 1 is " << result << endl;
```

```
    result = sum(1, 100);
```

```
    cout << "The sum from 1 to 100 step 1 is " << result << endl;
```

```
    result = sum(1, 100, 2);
```

```
    cout << "The sum from 1 to 100 step 2 is " << result << endl;
```

```
}
```



## 7.2 Default Arguments

- Default arguments can be **literals**, **constants** or **expressions**.

*int f( );*

*void delay(int k, int **time=f()**);*

- default arguments can be initialized *only once* in the *same scope*.

*void fun(int x=7);*

*void fun(int x=8);   //error: redefinition*



## 7.3 Function Overloading

- A mechanism for multiple functions with the same name, it is the reflection to polymorphism.
- In C++, a function is identified by not only the name but also the number , the types of the parameters and the keywords, const as member function of a class, which are called the signature.



# Examples

- void swap (unsigned long &, unsigned long &)
- void swap (double &, double &)
- void swap (char &, char &)
- void swap (Point &, Point &)

*They are different functions!!!*



# Choosing overloading vs. default arguments

- ◆ *efficiency*
- ◆ *interface*
- ◆ *understanding*

```
class Point
{
public:
    Point( )
    { coordX = 0; coordY = 0; }
    Point(double x, double y)
    { coordX = x; coordY = y; }
private:
    double coordX, coordY;
};
```

```
class Point
{
public:
    Point(double x = 0,
           double y = 0)
    { coordX = x;
      coordY = y; }
private:
    double coordX,
           coordY;
};
```



# summary

- **Function overloading**
- **Default arguments in the function**