

# 指针Pointers

变量指针与指针变量

Pointer of a variable

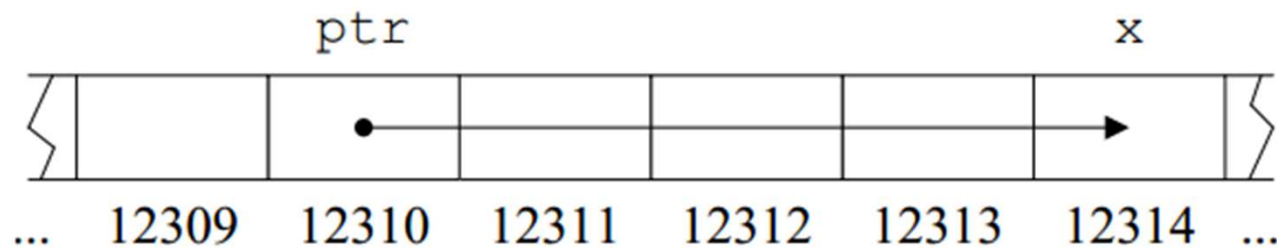
# 变量与内存(Variables and Memory)

- 当你声明一个变量时，计算机将给该变量一个内存，可以存储变量的值。
- 当你使用变量时，计算机将做两步操作：
  - 根据变量名查找其对应的地址；
  - 通过地址对该地址的变量内容进行读(retrieve)或写(set)。
- 变量的地址称为 变量的指针！
- C++通过运算符& 和\*对变量进行操作
  1. &x：得到变量的地址(指针)
  2. \*(&x)：先得到变量x的地址( &x ),然后根据地址得到其值( \*(&x) ).

&: 取地址  
\*: 取内容

# 指针(Pointers)

- 指针(Pointers)或者说地址(Memory addresses)使得操作数据更为方便。
  - 更灵活的传递引用;
  - 操纵复杂的数据, 即使他们是散乱分布的;
  - 多态性: 调用数据时不需要知道其类型
- 指针变量: 存储地址(指针)的变量。如果一个指针变量y存储的是另外变量x的地址, 则说变量y指向变量x. 经常形象化地用一个箭头表示这种指向!

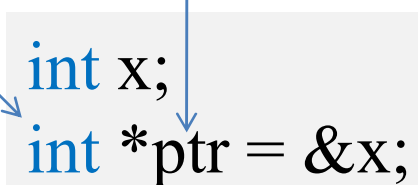


ptr存储的是x的地址

# 指针(变量)-声明

- 指针变量的定义格式:

```
data_type *pointer_name; // Add "= initial_value" if applicable
```

- 例如:  


```
int x;  
int *ptr = &x;
```

- 指针变量的使用:

```
*ptr = 30;           // *ptr就是ptr指向的那个变量  
std::cout<<ptr;      // 输出ptr存储的地址  
std::cout<<*ptr;     // 输出ptr指向的变量的值
```

## 指针(变量)-作为函数参数

- 将实参指针变量赋值给形参指针变量。如

```
void squareByPtr( int *numPtr ) {  
    *numPtr = *numPtr * *numPtr;  
}
```

```
int main() {  
    int x = 5;  
    squareByPtr(&x);  
    cout << x; // Prints 25  
}
```

指针变量numPtr的值是x的地址 (&x)

\*numPtr就是变量x

# 常指针(const Pointers)

- `const`关键字可以放在一个变量声明的两个位置，说明*指针本身或指针指向的变量*是不可以修改的常量。

```
const int *ptr;
```

// declares a changeable pointer to a constant integer.

```
int * const ptr;
```

// declares a constant pointer to changeable integer data.

```
const int * const ptr;
```

// forbids changing either the address ptr contains  
or the value it points to.

## Null, Uninitialized, and Deallocated Pointers

- 任何值为0的指针称为空指针(*null pointer*).  
空指针是一个无效(*invalid*)指针，访问任何指针指向的变量(称为解引用(*dereferencing*))时必须检查是否为空指针！
- 解引用一个不存在或销毁的变量会引起运行期错误！

```
int *myFunc() {  
    int phantom = 4;  
    return &phantom;  
}
```

- 当myFunc函数退出时，调用这个函数的其他函数不能使用无效的返回指针访问phantom！

# 引用(reference)

- 引用变量是其他变量的别名。定义引用变量的&不是取地址运算符！

```
int y;  
int &x = y; //引用变量x是y的别名
```

- 引用变量定义时必须初始化。
- 对引用变量的修改就是对被引用变量的修改。

```
int y = 3;  
int &x = y; //引用变量x是y的别名  
x = 10;  
std::cout<<y<<'\n';
```



# 引用(reference)

- 引用变量作为函数形式参数，对引用形参的修改就是对实际参数的修改！

```
void swap (int &x , int &y){  
    int t = x; x = y; y = t;  
}
```

```
void main (){  
    int a = 3, b = 5;  
    swap(a,b);  
    cout<<a <<" " <<b<<'\n';  
}
```

# \*和&的多面性

- \*在一个变量定义中时，说明这是定义一个指针变量。

```
int *p = &x;
```

- \*在一个定义好的指针变量前，表示解引用(dereferencing)这个指针变量。

```
*p = 34;
```

- &在一个变量定义中时，表示定义一个引用变量

```
int &y = x;    //y是x的别名
```

- &在一个定义好的变量前，表示获取这个变量的地址

```
int *p = &x;
```

# 指针与数组Pointers and Arrays

- 数组名实际上是数组第一个元素的地址。  
`myArray[3]`表示返回从第一个元素之后的第3个元素。
- 指针算术：通过对指针加减整数可以在不同地址之间移动。
- 指针移动的步长：由指针类型决定的。

```
long arr[] = {6, 0, 9, 6};
```

```
long *ptr = arr;
```

```
ptr++;
```

```
long *ptr2 = arr + 3;
```



ptr指向数组的第1个元素



ptr指向数组的第2个元素



ptr2指向数组的第4个元素

- `myArray[3]` 等价于 `*(myArray + 3)`

# 指针与数组Pointers and Arrays

- 指针除可以 加/减整数外，还可以进行两个指针的减法，以便获得两个指针的间隔元素的数目！

```
long arr[] = {6, 0, 9, 6};  
long *ptr = arr;  
ptr++;  
long *ptr2 = arr + 3;
```

```
int num = ptr2 - ptr; //两个指针之间间隔长整数的数目
```

## char \* 字符串

- 字符串就是字符数组。我们经常用字符型指针(char \*)指向这个字符串的第一个字符！

```
char str[] = { 'h' , 'e' , 'l' , 'l' , 'o' , '\0' };
```

```
char *str2 = str;
```

```
str2[2] = 'L';
```

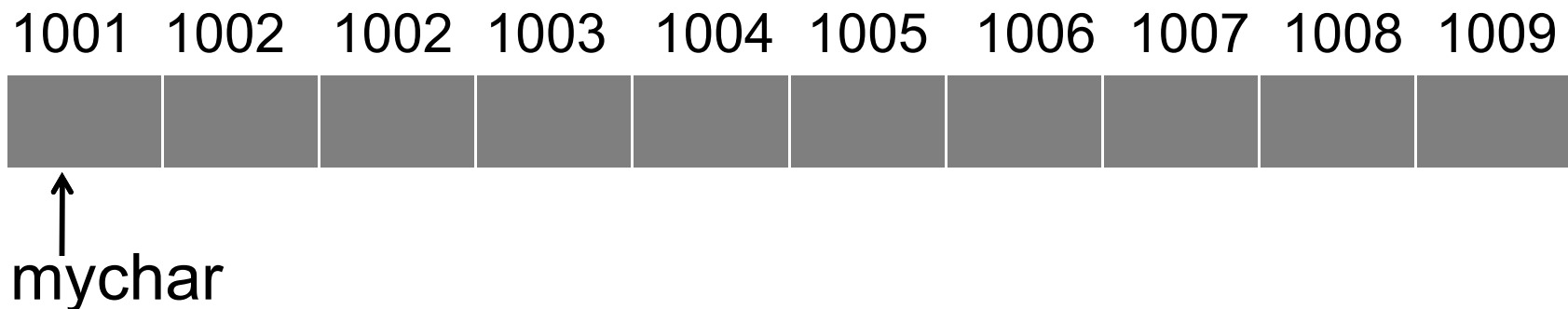
- 但不能修改一个字符指针指向的字符串常量，如不能修改下面的courseName2指向的“6.096”

```
char courseName1[] = { '6' , '.' , '0' , '9' , '6' , '\0' };  
char *courseName2 = "6.096";
```

# 指针的数学运算Arithmetic of pointers

- 指针只能进行加、减数学运算,但是指针的加法和减法的具体运算根据它所指向的数据的类型的的大小的不同而有所不同。
- 假设: 字符char 占用1 的字节(1 byte), 短整型short 占用2 个字节, 长整型long 占用4个字节。

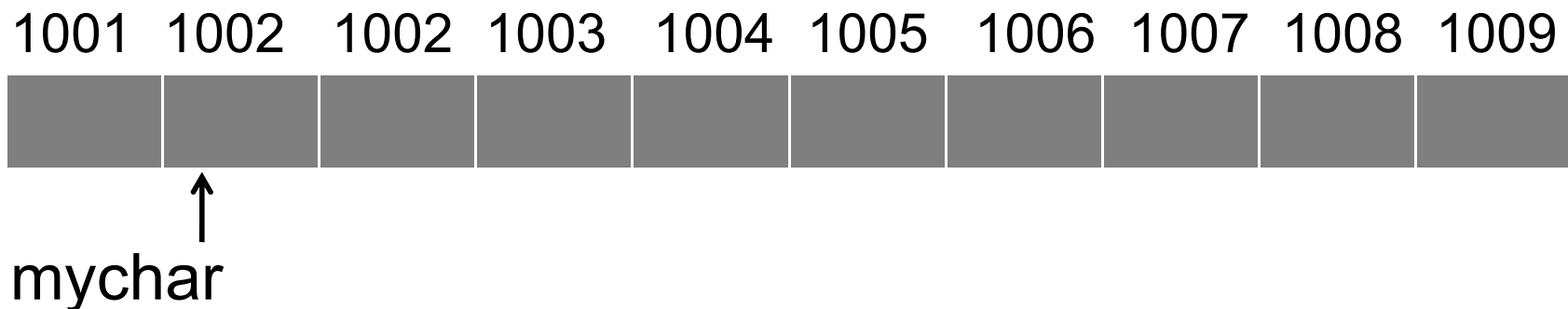
```
char *mychar;
```



# 指针的数学运算Arithmetic of pointers

- 指针只能进行加、减数学运算,但是指针的加法和减法的具体运算根据它所指向的数据的类型的的不同而有所不同。
- 假设: 字符char 占用1 的字节(1 byte), 短整型short 占用2 个字节, 长整型long 占用4个字节。

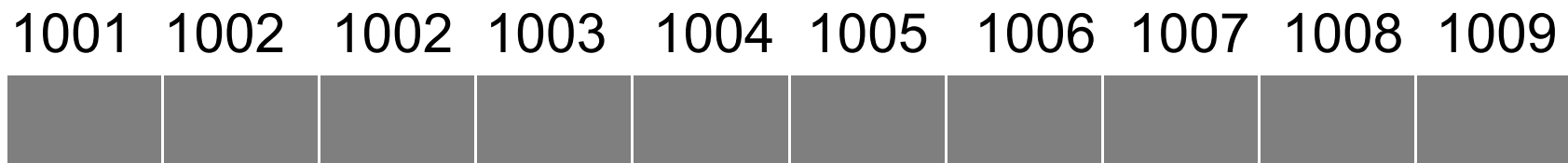
```
char *mychar;  
mychar++;
```



# 指针的数学运算Arithmetic of pointers

- 指针只能进行加、减数学运算,但是指针的加法和减法的具体运算根据它所指向的数据的类型的的不同而有所不同。
- 假设: 字符char 占用1 的字节(1 byte), 短整型short 占用2 个字节, 长整型long 占用4个字节。

```
char *mychar;  
mychar++;  
mychar = mychar+2;
```

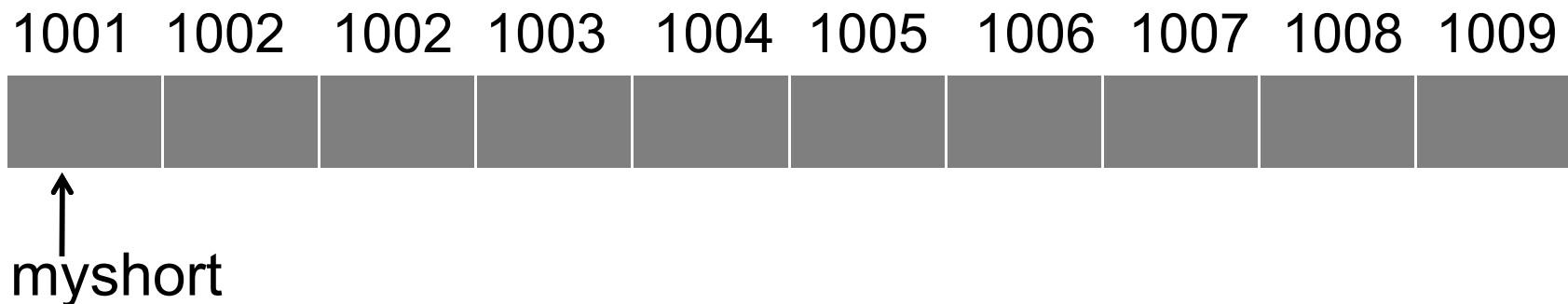




# 指针的数学运算Arithmetic of pointers

- 指针只能进行加、减数学运算,但是指针的加法和减法的具体运算根据它所指向的数据的类型的的不同而有所不同。
- 假设: 字符char 占用1 的字节(1 byte), 短整型short 占用2 个字节, 长整型long 占用4个字节。

```
short *myshort;
```



# 指针的数学运算Arithmetic of pointers

- 指针只能进行加、减数学运算,但是指针的加法和减法的具体运算根据它所指向的数据的类型的的不同而有所不同。
- 假设: 字符char 占用1 的字节(1 byte), 短整型short 占用2 个字节, 长整型long 占用4个字节。

```
short *myshort;  
myshort++;
```

1001 1002 1003 1004 1005 1006 1007 1008 1009 1010

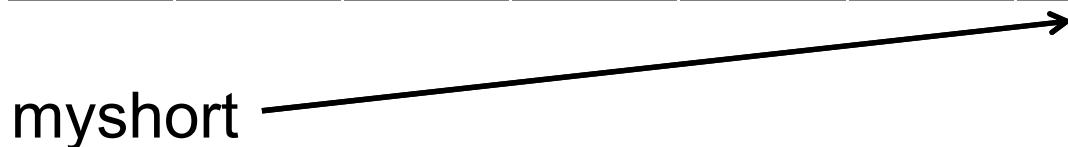
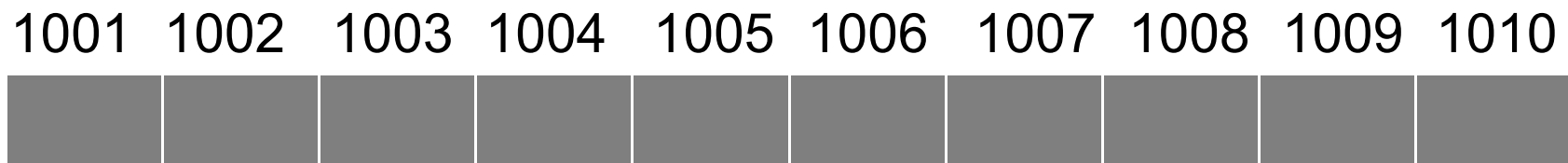


myshort →

# 指针的数学运算Arithmetic of pointers

- 指针只能进行加、减数学运算,但是指针的加法和减法的具体运算根据它所指向的数据的类型的的不同而有所不同。
- 假设: 字符char 占用1 的字节(1 byte), 短整型short 占用2 个字节, 长整型long 占用4个字节。

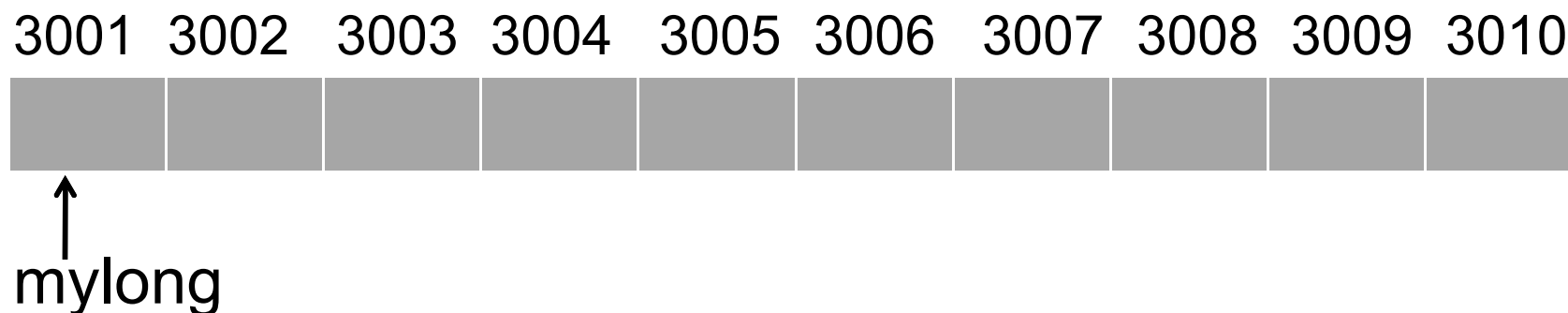
```
short *myshort;  
myshort++;  
myshort = myshort +2;
```



# 指针的数学运算Arithmetic of pointers

- 指针只能进行加、减数学运算,但是指针的加法和减法的具体运算根据它所指向的数据的类型的的大小的不同而有所不同。
- 假设: 字符char 占用1 的字节(1 byte), 短整型short 占用2 个字节, 长整型long 占用4个字节。

long \*mylong;



# 指针的数学运算Arithmetic of pointers

- 指针只能进行加、减数学运算,但是指针的加法和减法的具体运算根据它所指向的数据的类型的的不同而有所不同。
- 假设: 字符char 占用1 的字节(1 byte), 短整型short 占用2 个字节, 长整型long 占用4个字节。

```
long *mylong;
```

```
mylong++;
```

3001 3002 3003 3004 3005 3006 3007 3008 3009 3010

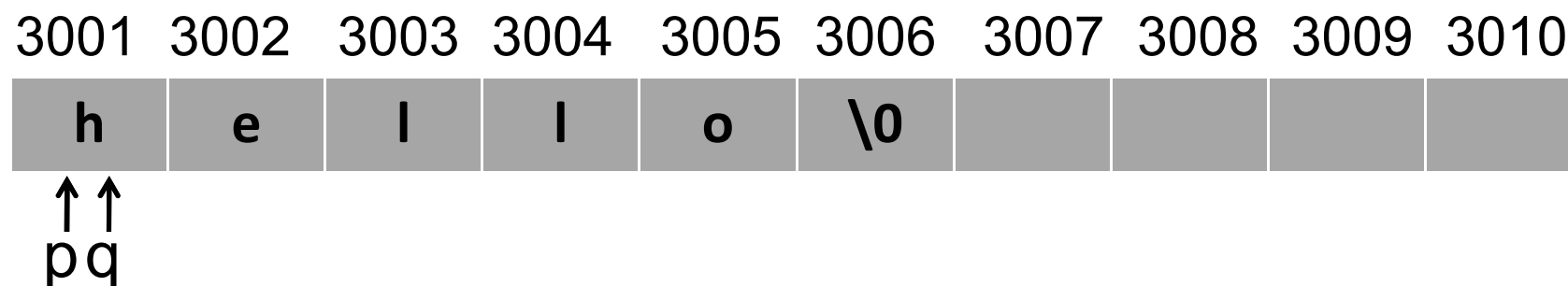


mylong →

# 指针的数学运算Arithmetic of pointers

- 指针只能进行加、减数学运算,但是指针的加法和减法的具体运算根据它所指向的数据的类型的不同的不同而有所不同。
- 假设: 字符char 占用1 的字节(1 byte), 短整型short 占用2 个字节, 长整型long 占用4个字节。

`char *p = "hello", *q = p;`



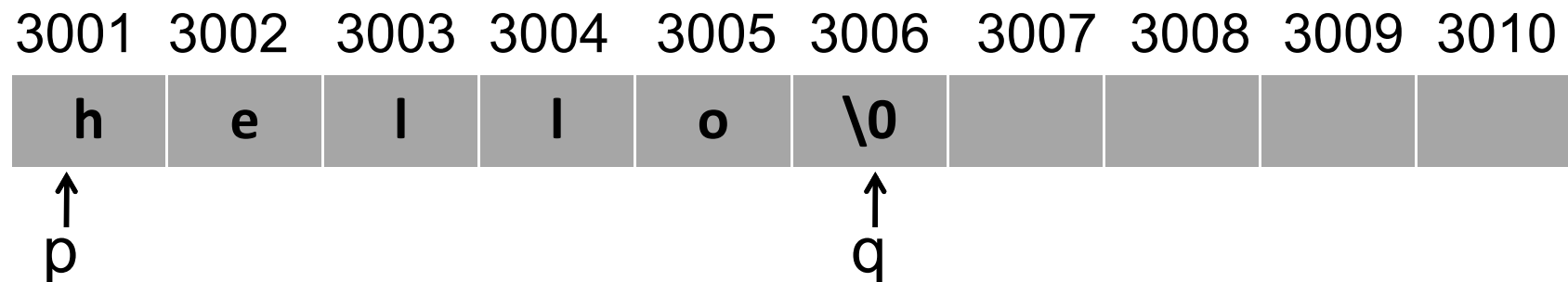
# 指针的数学运算Arithmetic of pointers

- 指针只能进行加、减数学运算,但是指针的加法和减法的具体运算根据它所指向的数据的类型的不同的不同而有所不同。
- 假设: 字符char 占用1 的字节(1 byte), 短整型short 占用2 个字节, 长整型long 占用4个字节。

```
char *p = "hello", *q = p;
```

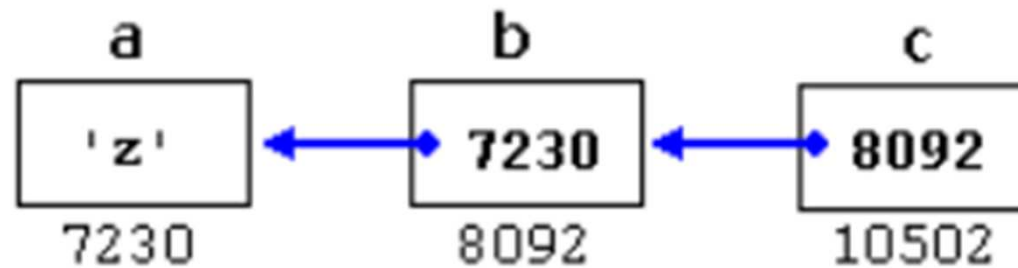
```
while(*q!='\0') q++;
```

```
int n = q - p; //5
```



# 指针的指针Pointers to pointers

```
char a;  
char * b;  
char ** c;  
a = 'z';  
b = &a;  
c = &b;
```



(方框内为变量的内容；方框下面为内存地址)

`c` 是一个(char \*\*)类型的变量，它的值是8092

`*c` 是一个(char\*)类型的变量，它的值是7230

`**c` 是一个(char)类型的变量，它的值是'z'



# 函数指针Pointers to functions

- 声明一个函数指针像声明一个函数原型一样,除函数指针名前要有\*, 并在括号()内。

```
type (*funPtr) ( parames );
```

```
int addition (int a, int b) {    return (a+b);  }  
int subtraction (int a, int b) {    return (a-b);  }  
int (*minus)(int,int) = subtraction;  
int main () {  
    int m = (* minus) (7,5) ;  
    minus = add;  
    int n = (* minus) (7,5) ;  
    cout <<m<<" "<<n<<"\n";  
    return 0;  
}
```

# 函数指针Pointers to functions

- 最大的作用是把一个函数作为参数传递给另外一个函数.

```
int addition (int a, int b) {    return (a+b);  }
int subtraction (int a, int b) {    return (a-b);  }
int (*minus)(int,int) = subtraction;
int operation (int x, int y, int (*functocall)(int,int)) {
    int g  = (*functocall)(x,y);    return (g);
}
int main () {
    int m = operation (7, 5, addition);
    int n = operation (20, m, minus);
    cout <<m<<" "<<n<<"\n";
    return 0;
}
```