第3章 字符串

In [1]:
```
#include <iostream>
#include <cstring>
#include <string>
using namespace std;
```

3.1 C中的字符串

字符串的初始化

首先不管c还是c++，随便写一串字符，这个字符串常量结尾自动有个字符'\0';

在c语言中，字符串通常用一维char数组保存，因此保存字符串时，数组大小要比真实的字符串长度大1

In [2]:
```
char s1[]="hello";
cout << s1 << endl;
cout << sizeof(s1) << endl;
```

```
hello
6
```

字符串的输入输出

cin和cout支持直接输入输出字符串，注意开辟一个足够大的数组防止越界

In [3]:
```
char s2[20];
cin >> s2;
cout << endl << s2;
```

```
world
```

字符串的访问

可以利用数组的[]操作符直接对某个字符进行修改

In [4]:
```
s2[0] = 'W';
cout << s2;
```

```
World
```

cstring中有关字符串处理的函数

In [5]:
```
strcmp(s1,s2);  //字符串比较，相等则返回0
```

Out[5]: 17

In [6]:
```
'h'-'W'
```

Out[6]: 17

In [7]:
```
char s3[20];
strcpy(s3,s1);  //字符串拷贝
cout << s3;
```

```
hello
```

In [8]:
```
strcat(s1,s2);
cout << s1;
```

```
helloWorld
```

void*指针与内存拷贝函数memcpy

In [9]:
```
float b;
unsigned int binValue = 0b10111110111000000000000000000000;  // 二进制数值
memcpy(&b, &binValue, sizeof(float));
cout << b;
```

```
-0.4375
```

上面memcpy的参数包含了两个不同类型的地址，这种通用的地址是如何做到的呢，就是void *指针*，

void 指针可以被任何类型的指针进行赋值，但是它没有取值，++，--等操作，我们利用这个特性手动实现以下memcpy；

In [10]:
```
void * myMemcpy(void *dst, void *src, int n){
    char *dst_ = (char *)dst;
    char *src_ = (char *)src;
    for (int i = 0; i < n; i++){
        dst_[i] = src_[i];
    }
    return dst_;
}
```

In [11]:
```
float b;
unsigned int binValue = 0b10111110111000000000000000000000;  // 二进制数值
myMemcpy(&b, &binValue, sizeof(float));
cout << b;
```

```
-0.4375
```

3.2 C++中的string类

初始化

In [12]:
```
string s1 = "hello world";
s1
```

Out[12]:
```
"hello world"
```

输入输出

注意遇到空格换行输入就结束

In [13]:
```
string s2;
cin >> s2;
cout << endl;
cout << s2;
```

```
zhang
```

赋值

In [14]:
```
s2=s1;
cout << s2;
```

```
hello world
```

下标运算符[]

In [15]:
```
s2[6] = 'W';
s2
```

Out[15]:
```
"hello World"
```

拼接

In [16]:
```
s1+s2
```

Out[16]:
```
"hello worldhello World"
```

大小比较

In [17]:
```
s1 < s2
```

Out[17]:
```
false
```

返回子串

In [18]:
```
s1.substr(1,3)
```

Out[18]:
```
"ell"
```

查找

In [25]:
```
s1
```

Out[25]:
```
"hello world"
```

In [26]:
```
s1.find("l")  //从左边开始找
```

Out[26]:
```
2
```

In [27]:
```
s1.rfind("l")  //从右边开始找
```

Out[27]:
```
9
```

In [28]:
```
s1.find("zhang")==string::npos  //找不到
```

Out[28]:
```
true
```

替换

In [30]:
```
s1.replace(1,2,"link") // 把s1[1:3]换成“lin”
```

Out[30]: "hlinklo world"

In [32]:
```
s1  //上面的操作是原地址操作
```

Out[32]: "hlinklo world"

插入

In [34]:
```
s1.insert(2,"WWW")
```

Out[34]: "hlWWWinklo world"

In [35]:
```
s1
```

Out[35]: "hlWWWinklo world"

删除

In [36]:
```
s1.erase(2,3)
```

Out[36]: "hlinklo world"

返回长度

In [37]:
```
s1.length()
```

Out[37]: 13

In [38]:
```
s1.size()
```

Out[38]: 13

In [ ]: