数组和字符串

标准库类型

vector

string

传统数组的缺点

大小必须在编译时确定，容易溢出

追加元素需要一个 i 记录最后一个位置的下标

不可复制和赋值

在 C 中，我们还可以通过在 heap 上开辟内存的方式来获取动态的数组，这样的好处是可以在运行期决定数组的大小，但是管理麻烦，如果忘了手工 free 内存，就会造成内存泄露。

vector

C++引入了新的数组 vector，可以克服上面的诸多缺点。

vector 的定义

直接定义一个空数组
也可以指定数组的大小
它可以无限的扩充，不存在溢出问题
支持的操作：
1） 下标访问
2） size（）
3） push_back 从数组的后面追加元素

vector 的使用方法如下：

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

int main() {

    vector<int> col;  //声明一个空数组

    for(int i = 1; i <= 6; ++i){
        col.push_back(i);//往数组中追加元素
    }

    //遍历打印
    for(int i = 0; i < col.size(); ++i){
        cout << col[i] << " ";
    }
    cout << endl;
}
```

这里我们采用的是声明一个空数组，后面给他追加元素的做法，事实上，这是一种常见的做法。还有一种用法是，一开始就制定好数组的大小，然后逐个进行赋值。

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

int main() {

    vector<int> col(20);  //数组大小为 20

    for(int i = 0; i != col.size(); ++i){
        col[i] = i;     //根据下标进行赋值
    }

    //遍历打印
    for(int i = 0; i < col.size(); ++i){
```

```
        cout << col[i] << " ";
    }
    cout << endl;
}
```

这里有一些注意点，就是下标不可越界访问，更不可越界赋值，标准中规定越界为非定义行为，引发的后果是未知的！

vector 的另外一种访问方式：迭代器

迭代器是一个“可遍历全部或者部分元素”的对象，但是它的表现行为像是一个指针。这里暂时不介绍迭代器的其他用法。用法如下：

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

int main() {

    vector<int> col(20);  //数组大小为 20

    for(int i = 0; i != col.size(); ++i){
        col[i] = i;     //根据下标进行赋值
    }

    for(vector<int>::iterator iter = col.begin(); iter != col.end(); ++iter){
        cout << *iter << " ";
    }
    cout << endl;
}
```

后面我们会接触到各种迭代器，但是他们的用法都大同小异。

这里有几处注意点：

对 vector 调用 begin，得到的是一个迭代器，指向 vector 的第一个元素。

对 vector 调用 end，返回一个迭代器，注意，它指向的是容器最后一个元素的下一个位置，也就是说，它指向的是一个不存在的位置。

于是 begin 和 end 就构成了一个半开区间，从第一个元素开始，到最后一个元素的下一个位置结束。半开区间有两个优点：

1.为“遍历元素时，循环的结束时机”提供了一个简单的判断依据。只要尚未到达 end，循环就可以继续进行。

2.不必对空区间采取特殊处理手法。空区间的 begin 等于 end

刚才使用迭代器是正向打印数组，如果想逆向呢，采用下标当然可以，迭代器呢？

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

int main() {

	vector<int> col(20);  //数组大小为 20

	for (int i = 0; i != col.size(); ++i) {
		col[i] = i;	//根据下标进行赋值
	}

	for (vector<int>::reverse_iterator iter = col.rbegin(); iter != col.rend();
			++iter) {
		cout << *iter << " ";
	}
```

```
    cout << endl;
}
```

这里跟刚才使用迭代器有几处区别：

1.使用的不再是 iterator，而是 reverse_iterator

2.调用的也不再是 begin 和 end，而是 rbegin 和 rend，注意 rbegin 指向最后一个元素，rend 指向第一个元素的前一个位置。

字符串：

C 风格字符串的缺陷：

1.恼人的结束符‘\0’

2.很多时候需要手工保证安全性，例如下面的代码就可能导致程序崩溃。

```
#include <iostream>
#include <string.h>
using namespace std;

int main() {

    char buf[3];
    strcpy(buf, "hello");
    cout << buf << endl;
}
```

这里错误的原因在于 buf 的内存空间不够，于是内存越界，但是 strcpy 不会也无法检查内存是否越界。

C++的解决方案是引入了 string。你可以把 string 当做一个一般的类型去使用，而不会发生任何问题。

C 风格字符串可进行的操作有

strlen 求长度

strcpy 复制字符串

strcmp 比较字符串大小

strcat 连接字符串

这些在 string 中均可以进行，而且更加简单，没有任何的安全隐患。

string 用法如下：

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

int main() {

    string s1 = "hello";
    string s2("test");
    string s3;      //空字符串

    cout << s1 << endl; //打印字符串
    cout << s1.size() << endl; //求长度
    s3 = s1;    //字符串间的复制
    s3 += s2;      //字符串的拼接
    cout << s3 << endl;

    //比较大小
```

```
        cout << (s1 < s2) << endl;
        cout << (s1 == s3) << endl;
    }
```

可以看出获取字符串大小使用 string 内部的 size 方法，其他操作采用正常的运算符就可以。

关于相加，这里有几点：

1.string 和 string 可以相加

2.string 和 char*可以相加

3.char*和 string 可以相加

4.char*和 char*不可以相加!!

自行写程序验证

string 同样支持下标操作，所以可以采用下标遍历：

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

int main() {

    string s = "who are you?";

    for (size_t ix = 0; ix != s.size(); ++ix) {
        cout << s[ix] << " ";
    }
    cout << endl;
}
```

同样支持迭代器 iterator 和逆置迭代器 reverse_iterator

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

int main() {

	string s = "who are you?";

	for (string::iterator iter = s.begin(); iter != s.end(); ++iter) {
		cout << *iter << " ";
	}
	cout << endl;

	for (string::reverse_iterator iter = s.rbegin(); iter != s.rend(); ++iter) {
		cout << *iter << " ";
	}
	cout << endl;
}
```

在 string 中查找字符：

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

int main(int argc, char **argv) {
	string s = "American";

	string::size_type pos = s.find('i');
	if (pos != string::npos) { //查找成功
		cout << pos << endl;
	} else {                        //没有找到
		cout << "not found!" << endl;
	}
}
```

使用的 find 函数，这里要对查找结果进行判断。

string 可以转化为 C 风格字符串：

数组和字符串　郭春阳

```
string s = "Shenzhen";
cout << s.c_str() << endl;
```

注意：string 得出的 char*是 const 属性，也就是说只可以读取它的值，不可以对其进行改动。

另外这个值可能失效，所以如果需要操作字符串，最好是复制一份。

如何整行读入字符串：

使用 getline

代码如下：

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

int main() {
	string s;
	getline(cin, s);
	cout << s << endl;
}
```