/\*

\* STL六大组件：

\* 容器：各种数据结构，如vector，list，deque，set，map等，用来存放数据

\* 算法：各种常用的算法：如find，sort，copy，for\_each等。

\* 迭代器：扮演了容器与算法之间的胶合剂。

\*

\* 仿函数：行为类似函数，可作为算法的某种策略。

\* 适配器：一种用来修饰容器或迭代器接口的东西。

\* 空间配置器：负责空间的配置与管理。

\*/

// string基本概念

/\*

\*本质： string是C++风格字符串，而string本质桑是一个类

\*string和char\*的区别：

\* char\* 是一个字符串

\* string 是一个类，类内封装了一个char\*，管理这个字符串，是一个char\*容器

\*

\* 特点：

\* string 类内封装了很多成员方法

\* 例如：查找find 拷贝copy 删除delete 替换replace 插入insert

\* string管理char\*所分配的内存，不用担心复制越界和取值越界等，由类内进行负责

\*/

/\*

\*构造函数原型：

\* string(): //构造一个空的字符串 例如：string str;

\* string(const char\* s); //使用字符串s初始化

\* string(const string& str); //使用一个string对象初始化另一个string对象

\* string(int n, char c); //使用n个字符串初始化

\*/

#include<iostream>

#include<string>

using namespace std;

void test1()

{

string s1; //默认构造

const char\* str = "hello word";

string s2(str);

cout << "s2 = " << s2 << endl;

string s3(s2);

cout << "s3 = " << s3 << endl;

string s4(10, 'a');

cout << "s4 = " << s4 << endl;

}

int main()

{

test1();

return 0;

}

// string赋值操作

/\*

\*函数原型：

\* string& operator=(const char\* s); //char\*类型字符串 赋值给当前字符串

\* string& operator=(const string &s); //把字符串s赋给当前的字符串

\* string& operator=(char c); //字符赋值给当前字符串

\* string& assign(const char \*s); //把字符串s赋值给当前字符串

\* string& assign(const char \*s,int n); //把字符串s的当前n个字符赋给当前的字符串

\* string& assign(const string &s); //把字符串s赋值给当前字符串

\* string& assign(int n, char c); //用n个字符c赋给当前字符串

\*/

#include<iostream>

#include<string>

using namespace std;

void test1()

{

string str1;

str1 = "hello world";

cout << "str1 = " << str1 << endl;

string str2 = str1;

cout << "str2 = " << str2 << endl;

string str3;

str3 = 'a';

cout << "str3 = " << str3 << endl;

string str4;

str4.assign("hello cpp");

cout << "str4 = " << str4 << endl;

string str5;

str5.assign("hello\_cpp",5);

cout << "str5 = " << str5 << endl;

//输出： hello //将字符串"helle\_cpp" 的前五个字符输出

string str6;

str6.assign(str5);

cout << "str6 = " << str6 << endl;

string str7;

str7.assign(10, 'w');

cout << "str7 = " << str7 << endl;

//输出： wwwwwwwwww

}

int main()

{

test1();

return 0;

}

// stirng字符串的拼接

//功能： 实现在字符串末尾拼接字符串

/\*

\* 函数原型：

\* string& operator+=(const char\* str); //重载+=操作符

\* string& operator+=(const char c); //重载+=操作符

\* string& operator+=(const str& str); //重载+=操作符

\* string& append(const char\* s); //把字符串s连接到当前字符串结尾

\* string& append(const char\* s, int n); //把字符串s的前n个字符连接到当前字符串结尾

\* string& append(const string &s); //同operator+=(const string& str)

\* string& append(const string &s,int pos,int n; //字符串s中从pos开始的n个字符连接到字符串结尾

\*/

#include<iostream>

#include<string>

using namespace std;

void test1()

{

string str1 = "我";

str1 += "爱学习";

cout << "str1 = " << str1 << endl;

str1 += ':';

cout << "str1 = " << str1 << endl;

string str2 = "C++ 算法";

str1 += str2;

cout << "str1 = " << str1 << endl;

string str3 = "I";

str3.append(" love ");

cout << "str3 = " << str3 << endl;

str3.append("study play ", 5);

cout << "str3 = " << str3 << endl;

str3.append(str2);

cout << "str3 = " << str3 << endl;

str3.append(str2, 4, 6);

cout << "str3 = " << str3 << endl;

//从第4个截取到第5个

}

int main()

{

test1();

return 0;

}

// 查找和替换

#include<iostream>

#include<string>

using namespace std;

//查找

void test1()

{

string str1 = "abcdefg";

//find从左往右查找str第一次出现的位置，从pos默认为0开始查找

int pos = str1.find("de");

if (pos == -1) cout << "未找到" << endl;

else cout << "pos = " << pos << endl;

//rfind从右往左查

pos = str1.rfind("ef");

if (pos == -1) cout << "未找到" << endl;

else cout << "pos = " << pos << endl;

}

//替换

void test2()

{

string str1 = "abcdefg";

str1.replace(1, 3, "apple");

cout << "str1 = " << str1 << endl;

//从1位置起3个字符替换为apple

}

int main()

{

test1();

test2();

return 0;

}

// 字符串的比较

/\*

\* 按照ASCII码进行比较

\* = 返回 0

\* > 返回 1

\* < 返回 -1

\*/

#include<iostream>

#include<string>

using namespace std;

void test1()

{

string str1 = "hello";

string str2 = "hello";

if (!str1.compare(str2)) cout << "str1 等于 str2" << endl;

}

int main()

{

test1();

return 0;

}

// string字符存取

#include<iostream>

#include<string>

using namespace std;

void test1()

{

string str1 = "hello";

cout << "str1 = " << str1 << endl;

//1. 通过[]访问单个字符

for (int i = 0; i < str1.size(); i++) cout << str1[i] << " ";

//size可获取str1的长度

cout << endl;

//2. 通过at方式访问单个字符

for (int i = 0; i < str1.size(); i++) cout << str1.at(i) << " ";

cout << endl;

//修改单个字符

str1[0] = 'H';

cout << "str1 = " << str1 << endl;

str1.at(0) = 'h';

cout << "str1 = " << str1 << endl;

}

int main()

{

test1();

return 0;

}

// 插入和删除

#include<iostream>

#include<string>

using namespace std;

void test1()

{

string str1 = "hello ";

//插入

str1.insert(5,"11111");

cout << "str1 = " << str1 << endl;

//删除

str1.erase(5, 5);

cout << "str1 = " << str1 << endl;

}

int main()

{

test1();

return 0;

}

// string子串

#include<iostream>

#include<string>

using namespace std;

//求子串

void test1()

{

string str1 = "hello world";

string subStr = str1.substr(6, 5);

cout << "subStr = " << subStr << endl;

}

//案例： 截取姓名

void userName() {

string e1 = "zhangsan@qq.com";

string e2 = "lisi@qq.com";

string e3 = "wangwu@qq.com";

//从邮件中 获取 用户信息

int pos = e1.find("@");

string userName = e1.substr(0, pos);

cout << userName << endl;

pos = e2.find("@");

userName = e2.substr(0, pos);

cout << userName << endl;

pos = e3.find("@");

userName = e3.substr(0, pos);

cout << userName << endl;

}

int main()

{

test1();

userName();

return 0;

}

// vector容器

//功能： vector数据结构和数组非常相似，也称单端数组

//区别： vector可以动态扩展

#include<iostream>

#include<vector>

using namespace std;

void printVector(vector<int>& v)

{

for (vector<int>::iterator i = v.begin(); i != v.end(); i++) cout << \*i << " ";

cout << endl;

}

void test1()

{

//默认构造 无参构造

vector<int>v1;

for (int i = 0; i < 10; i++) v1.push\_back(i);

printVector(v1);

//通过区间方式进行构造

vector<int>v2(v1.begin(), v1.end());

printVector(v2);

//n个elem方式进行构造

vector<int>v3(10, 100);

printVector(v3);

}

int main()

{

test1();

return 0;

}

// vector赋值操作

#include<iostream>

#include<vector>

using namespace std;

void printVector(vector<int>& v)

{

for (vector<int>::iterator i = v.begin(); i != v.end(); i++) cout << \*i << " ";

cout << endl;

}

void test1()

{

vector<int>v1;

for (int i = 0; i < 10; i++) v1.push\_back(i);

printVector(v1);

//赋值 operator=

vector<int>v2;

v2 = v1;

printVector(v2);

//assign

vector<int>v3;

v3.assign(v1.begin(), v1.end());

printVector(v3);

//n个elem方式

vector<int>v4;

v4.assign(10, 100);

printVector(v4);

}

int main()

{

test1();

return 0;

}

// vector容量和大小

#include<iostream>

#include<vector>

using namespace std;

void printVector(vector<int>& v)

{

for (vector<int>::iterator i = v.begin(); i != v.end(); i++) cout << \*i << " ";

cout << endl;

}

void test1()

{

vector<int>v1;

for (int i = 0; i < 10; i++) v1.push\_back(i);

printVector(v1);

if (v1.empty()) cout << "v1为空" << endl;

else

{

cout << "v1不为空" << endl;

cout << "v1的容量为： " << v1.capacity() << endl;

cout << "v1的大小： " << v1.size() << endl;

}

//重新指定大小

v1.resize(13); //默认填充为0

printVector(v1);

v1.resize(15, 100); //如果重新指定大小比原来小，超出部分会删除掉

printVector(v1);

}

int main()

{

test1();

return 0;

}

// vector插入和删除

/\*

\* 函数原型：

\* push\_back(ele); //尾部插入元素

\* pop\_back(); //删除最后一个元素

\* insert(const\_iterator pos, ele); //迭代器指定位置pos插入元素ele

\* insert(const\_iterator pos, int count, ele); //迭代器指定位置pos插入count个元素饿了、

\* erase(const\_iterator pos); //删除迭代器指向的元素

\* erase(const\_iterator start, const\_iterator end); //删除迭代器从start到end之间的元素

\* clear(); //删除容器中的所有元素

\*/

#include<iostream>

#include<vector>

using namespace std;

void printVector(vector<int>& v)

{

for (vector<int>::iterator i = v.begin(); i != v.end(); i++) cout << \*i << " ";

cout << endl;

}

void test1()

{

vector<int>v1;

//尾插

for (int i = 10; i <= 50; i += 10) v1.push\_back(i);

printVector(v1);

//尾删

v1.pop\_back();

printVector(v1);

//插入 (尾插）

v1.insert(v1.end(), 100);

printVector(v1);

v1.insert(v1.end() - 1, 2, 1000);

printVector(v1);

//删除

v1.erase(v1.begin());

printVector(v1);

v1.erase(v1.begin(), v1.end());

printVector(v1);

//类似于清空 即v1.clear();

v1.clear();

}

int main()

{

test1();

return 0;

}

// vector容器 数据存取

#include<iostream>

#include<vector>

using namespace std;

void test1()

{

vector<int>v1;

for (int i = 0; i < 10; i++) v1.push\_back(i);

//利用[]方式访问数组

for (int i = 0; i < v1.size(); i++) cout << v1[i] << " ";

cout << endl;

//利用at方式访问元素

for (int i = 0; i < v1.size(); i++) cout << v1.at(i) << " ";

cout << endl;

//获取首尾元素

cout << v1.front() << endl;

cout << v1.back() << endl;

}

int main()

{

test1();

return 0;

}

// vector容器 互换

#include<iostream>

#include<vector>

using namespace std;

void printVector(vector<int>& v)

{

for (vector<int>::iterator i = v.begin(); i != v.end(); i++) cout << \*i << " ";

cout << endl;

}

void test1()

{

vector<int>v1;

for (int i = 0; i < 10; i++) v1.push\_back(i);

vector<int>v2;

for (int i = 10; i < 120; i+=10) v2.push\_back(i);

//交换前

cout << "交换前：" << endl;

printVector(v1);

printVector(v2);

//交换后

v1.swap(v2);

cout << "交换后：" << endl;

printVector(v1);

printVector(v2);

}

//实际用途

//巧用swap可以收缩内存空间

void test2()

{

vector<int>v;

for (int i = 0; i < 100000; i++) v.push\_back(i);

cout << "v的容量： " << v.capacity() << endl;

cout << "v的大小： " << v.size() << endl << endl;

//重定大小

//resize之后，大小改变但容量不变，占用的内存也不变

v.resize(3);

cout << "v的容量： " << v.capacity() << endl;

cout << "v的大小： " << v.size() << endl << endl;

//用swap()收缩内存

vector<int>(v).swap(v);

cout << "v的容量： " << v.capacity() << endl;

cout << "v的大小： " << v.size() << endl;

//vector<int>(v)//匿名对象 当前行执行完之后被回收

}

int main()

{

test1();

test2();

return 0;

}

// vector容器 预留空间

//减少vector在动态扩展容量的次数

//函数原型： reserve(int len); //容器预留len个元素长度， 预留位置不初始化，元素不可访问

#include<iostream>

#include<vector>

using namespace std;

void printVector(vector<int>& v)

{

for (vector<int>::iterator i = v.begin(); i != v.end(); i++) cout << \*i << " ";

cout << endl;

}

void test1()

{

vector<int>v;

//利用reserve预留空间

v.reserve(100000);

int num = 0; //统计开辟次数

int\* p = NULL;

//测试开空间次数

for (int i = 0; i < 100000; i++)

{

v.push\_back(i);

if (p != &v[0])

{

p = &v[0];

num++;

}

}

cout << "num = " << num << endl;

}

int main()

{

test1();

return 0;

}