关联容器

概述：

目录：

Pair 类型

Pair 是一种简单的关联类型

创建/初始化/操作如下：

```
#include "test_include.h"

using namespace std;


int main(int argc, char **argv) {

	std::pair<int, string> p1;	// default
	std::pair<int, int> p2(4, 5);
	std::pair<string, string> p3("hello", "world");
	p1.first = 1;
	p1.second = "test";

	cout << p2.first << " " << p2.second << endl;
	cout << p3.first << " " << p3.first << endl;
	make_pair(1, 2);

	make_pair(string("hello"), 1);	// string , int
	vector<string> v1;
	vector<list<string> > v2;
	make_pair(v1, v2);
}
```

Map 类型

Map 可以看做是一种 pair 的容器，内部时采用二叉树实现的（具体些是红黑树）

Map 的定义方式：

```
#include "test_include.h"
#include <stack>
#include <queue>
using namespace std;

class Student
{
private:
	int _num;
	string _name;
};

int main(int argc, char **argv) {

	// map    key    value

	std::map<int ,int> m1;

	std::map<string, int> m2;
	map<string, string> m3;


	map<string, vector<string> > m4;

	map<list<vector<list<string> > >, stack<queue<int> > > m5;

	map<int, Student> m6;
	map<Student, int> m7;
}
```

Map 的遍历

```
#include "test_include.h"
#include <stack>
#include <queue>
```

```
#include <typeinfo>
using namespace std;

//void print(const map<string, int>::value_type &p)
void print(const pair<string, int> &p)
{
	cout << p.first << " " << p.second << endl;
}

int main(int argc, char **argv) {
	map<string, int> people;

	people["shenzhen"] = 1000;
	people["beijing"] = 3000;
	people["shanghai"] = 2000;


	cout << people["beijing"] << endl;

	people["beijing"] = 8000;    // OK

	//遍历这个 map
	map<string, int>::iterator iter = people.begin();
	while (iter != people.end()) {
		cout << iter->first << " " << iter->second << endl;
		++iter;
	}

	cout << "----------" << endl;

	for_each(people.begin(), people.end(), print); //这里用的是标准库算法
}
```

Map 使用下标访问的一些问题

```
	#include "test_include.h"
	#include <stack>
	#include <queue>
	#include <typeinfo>
	using namespace std;

	int main(int argc, char **argv) {
		map<string, int> word_count;
```

```
    cout << word_count.size() << endl;    // 0

    word_count["hello"];

    cout << word_count.size() << endl; // 1

    word_count["hello"];

    cout << word_count.size() << endl;    //1

    word_count["world"];

    cout << word_count.size() << endl;    //2
}
```

每当用下标去访问 map 元素的时候，如果该元素不存在，那么首先在 map 中新生成一个键值对。所以用下标访问不存在的键值对，会增加容器的大小

利用这一特性，可以实现一个单词计数程序：

```
#include "test_include.h"
using namespace std;

void print(const map<string, int>::value_type &p) {
    cout << p.first << " occurs : " << p.second << " times" << endl;
}

int main(int argc, char **argv) {
    map<string, int> word_count;
    string word;

    while (cin >> word) {
        word_count[word]++;    //
```

```
    }

    for_each(word_count.begin(), word_count.end(), print);

}
```

在 map 中添加元素

刚才看到，采用下标的方式，可以给 map 添加元素，但更好的做法时采用 insert 插入一个 pair 对象。

```
#include "test_include.h"
using namespace std;

void print(const map<string, int>::value_type &p) {
    cout << p.first << " occurs : " << p.second << " times" << endl;
}

int main(int argc, char **argv) {
    map<string, int> word_count;
    string word;

    word_count.insert(map<string, int>::value_type("hello", 1));
    for_each(word_count.begin(), word_count.end(), print);
    cout << "--------------" << endl;
    word_count.insert(make_pair("test", 3));
    for_each(word_count.begin(), word_count.end(), print);
    cout << "----------------" << endl;

    pair<map<string, int>::iterator, bool> ret = word_count.insert(
            map<string, int>::value_type("hello", 4));
    for_each(word_count.begin(), word_count.end(), print);
    cout << "----------------" << endl;

    cout << ret.first->first << endl;      // hello
```

```
    cout << ret.first->second << endl;
    cout << ret.second << endl;       //

}
```

这里值得注意的是 insert 的返回值，返回了一个 pair 对象，第一个元素是指向该 key 的迭代器，第二个则表示插入是否成功

利用这个 insert，我们改写一些刚才的单词计数程序：

```
#include "test_include.h"
#include <stack>
#include <queue>
#include <typeinfo>
using namespace std;

void print(const map<string, int>::value_type &p) {
    cout << p.first << " occurs : " << p.second << " times" << endl;
}

int main(int argc, char **argv) {
    map<string, int> word_count;
    string word;

    while (cin >> word) {
        std::pair<std::map<std::string, int>::iterator, bool> ret =
                word_count.insert(
                        std::map<std::string, int>::value_type(word, 1));
        if(!ret.second)
        {
            ++ret.first->second;
        }
    }

    for_each(word_count.begin(), word_count.end(), print);

}
```

这里有必要解释++ret.first->second 这行语句

Ret 是个 pair 对象

Ret.first 表示插入元素的迭代器

Ret.first->second 表示该键值对中的 value，也就是单词出现的个数

在 map 中查找元素

刚才看到可以利用下标获取 value 的值，但是这样存在一个弊端，如果下标访问的是不存在的元素，那么会自动给 map 增加一个键值对，这显然不是我们所预期的。

采用 count 和 find 来解决问题

Count 仅仅能得出该元素是否存在，而 find 能够返回该元素的迭代器

```
#include "test_include.h"
#include <stack>
#include <queue>
#include <typeinfo>
using namespace std;

void print(const map<string, int>::value_type &p) {
    cout << p.first << " occurs : " << p.second << " times" << endl;
}

int main(int argc, char **argv) {
    map<string, int> word_count;
    string word;
    word_count["test"] = 10;
    word_count["foo"] = 5;
    word_count["bar"] = 12;
```

```
    cout << word_count.count("test") << endl;      //count 0 or 1

    cout << word_count.count("hello") << endl;

    map<string, int>::iterator iter = word_count.find("test");
    if (iter == word_count.end()) {
        cout << "not found" << endl;
    } else {
        cout << iter->first << " " << iter->second << endl;
    }

    for_each(word_count.begin(), word_count.end(), print);
    cout << "----------" << endl;
//  word_count.erase(iter);
    word_count.erase("test");
    for_each(word_count.begin(), word_count.end(), print);

}
```

Set 容器

Set 类似于数学上的集合，仅仅表示某个元素在集合中是否存在，而不必关心它的具体位置。同样，set 中的元素互异，也就是无法两次插入相同的元素

使用方式和 map 类似，但是简单很多

```
#include "test_include.h"
using namespace std;



int main(int argc, char **argv) {
    set<int> s;

    for (size_t ix = 0; ix != 10; ++ix) {
```

```
            s.insert(ix);
            s.insert(ix);
        }

        cout << s.size() << endl; // 10

        s.insert(12);
        cout << s.size() << endl; // 11
        s.insert(12);
        cout << s.size() << endl;    //11

}


#include "test_include.h"
using namespace std;

void print(const string &s) {
        cout << s << " ";
}

int main(int argc, char **argv) {
        set<string> s;

        s.insert("hello");
        s.insert("world");
        s.insert("test");
        s.insert("foo");
        s.insert("bar");

        for_each(s.begin(), s.end(), print);
        cout << endl;

        s.erase("test");
        for_each(s.begin(), s.end(), print);
        cout << endl;

        set<string>::iterator iter = s.find("bar");
        if (iter == s.end()) {
                cout << "404" << endl;
        } else {
                s.erase(iter);
        }
        for_each(s.begin(), s.end(), print);
```

```
    cout << endl;

}
```

可以采用 set 实现一个停用词功能，在我们刚才的单词统计程序中，并不一定要统计所有的单词，而是要排除掉一部分

```
#include "test_include.h"
#include <fstream>
#include <stdexcept>
using namespace std;

ifstream &open_file(ifstream &is, const string &filename) {
    is.close();
    is.clear();
    is.open(filename.c_str());
    return is;
}

void restricted_wc(ifstream &input_file, const set<string> &exclude_words,
        map<string, int> &word_count) {
    string word;
    while (input_file >> word) {
        if (!exclude_words.count(word)) {
            ++word_count[word];
        }
    }
    input_file.close();
    input_file.clear();
}

void print(const string &s) {
    cout << s << " ";
}

void print_map(const map<string, int>::value_type &p) {
    cout << p.first << " occurs " << p.second << " times" << endl;
}

int main(int argc, char **argv) {
    set<string> exclude_words;
```

```
    map<string, int> word_count;

    exclude_words.insert("the");
    exclude_words.insert("a");
    exclude_words.insert("an");
    exclude_words.insert("I");
    exclude_words.insert("and");

    std::ifstream infile;
    string name = "in.txt";
    if (!open_file(infile, name)) {
        throw std::runtime_error("open file error!");
    }

    restricted_wc(infile, exclude_words, word_count);

    for_each(exclude_words.begin(), exclude_words.end(), print);
    cout << endl << "----------" << endl;

    for_each(word_count.begin(), word_count.end(), print_map);
    cout << endl;
}
```

STL 算法