**本示例使用的头文件**

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

#include <functional>

#include <list>

#include <iterator>

**基本函数**

vector<int> numList { 1, 2, 3, 4 };

// 查找值，范围 numList.begin() - numList.end() ，不包括 numList.end()

vector<int>::iterator a = find(numList.begin(), numList.end(), 2); // 返回指向2的迭代器，如果未找到，返回 numList.end()

// 求和，范围 numList.begin() - numList.end()，初始值0，需要类型实现 + 运算符

accumulate(numList.begin(), numList.end(), 0);

// 比较 equal1 的数量应该多余或等于 numList

vector<int> equal1 { 1, 2, 3, 4, 5};

equal(numList.begin(), numList.end(), equal1.begin());

// 填充

fill(numList.begin(), numList.end(), 0);

fill\_n(numList.begin(), 4, 0);

// 插入迭代器，back\_inserter返回一个插入迭代器（插入迭代器重载的赋值运算符）

auto it = back\_inserter(numList);

\*it = 5; // 插入 5 到numList

it++; // 增加迭代器

\*it = 6; // 插入 6

for(auto i : numList){

cout << i << endl;

}

// 拷贝 numList 的元素到 copy1

vector<int> copy1;

copy(numList.begin(), numList.end(), copy1.begin());

// 替换

replace(numList.begin(), numList.end(), 0, 10); // 将 0 替换成 10

vector<int> replace1;

replace\_copy(numList.begin(), numList.end(), replace1.begin(), 0, 10); // 将 0 替换成 10，放到replace1数组中

// 排序

sort(numList.begin(), numList.end());

// 去重

auto end\_unique = unique(numList.begin(), numList.end()); // 返回不重复区域的后一个位置

numList.erase(end\_unique, numList.end()); // 移除重复区域

**lambda表达式**

// [s] 为捕获列表，这样 s 就可以在lambda内部使用，这里 s 使用值类型，所以在传入lambda是固定为 "a"，要使用引用类型请使用 &s

// (const string &left, const string &right) 为参数列表

string s = "a";

auto lambda1 = [s](const string &left, const string &right){

cout << s << endl;

return left.size() < right.size();

};

lambda1("a", "b");

// 隐式捕获，[=]表示采用值类型，如果要使用引用请使用 [&]

auto lambda2 = [=](const string &left, const string &right){

cout << s << endl;

return left.size() < right.size();

};

**bind函数**

void bindFunTest(string param1, int param2){

cout << param1 << endl;

cout << param2 << endl;

}

void bindTest(){

// bindFunTest需要两个参数，按顺序绑定 placeholders::\_1 和 6 到 bindFunTest，生成只需要一个参数的函数

auto fun1 = bind(bindFunTest, std::placeholders::\_1, 6);

// "a" 为 placeholders::\_1

fun1("a");

// 1 为 std::placeholders::\_1， "abd" 为 std::placeholders::\_2

auto fun2 = bind(bindFunTest, std::placeholders::\_2, std::placeholders::\_1);

fun2(1, "abd");

string s = "a";

// 绑定 s，这里 s 为拷贝方式，引用方式请使用 ref(s)

auto fun3 = bind(bindFunTest, s, 6);

fun3();

}

**泛型算法使用自定义操作**

vector<string> list { "ab", "cd", "ef" };

// 使用 lambda 自定义比较操作

find\_if(list.begin(), list.end(), [](const string &value){

return value == "cd";

});

**算法迭代器**

list<string> strList { "ab", "cd", "ef" };

// 插入迭代器

back\_inserter(strList); // 返回尾部插入迭代器

front\_inserter(strList); // 返回前面插入迭代器

inserter(strList, strList.end()); // 返回插入迭代器，在strList.end()指定的位置位置前面插入

// 流迭代器，位于头文件 #include <iterator>，std命名空间

// 读取流迭代器，用来读取流和写入流的迭代器

istream\_iterator<int> intIt(cin); // 读取流迭代器，这里由于使用的是cin流，所以程序会等待终端输入

istream\_iterator<int> intEOF; // 尾后迭代器

while(intIt != intEOF){ // 输入eof可退出循环，window eof 按 ctrl+z

cout << \*intIt << endl;

intIt++; // 递增流迭代器，这里由于使用的是cin，如果没有数据，程序会等待终端输入

}

// 写入流迭代器

ostream\_iterator<string> strOt(cout); // 写入流迭代器

for(auto s : strList){

\*strOt = s; // 这会在 cout 输出 s

strOt++;

}

**算法参数模式**

如下是泛型算法的基本模式

Alg(begin, end, other args);

Alg(begin, end, dest, other args);

Alg(begin, end, begin2, other args);

Alg(begin, end, begin2, end2, other args);

有些算法有支持谓词操作的重载（自定义操作），谓词一般是最后一个参数

Alg(begin, end, opt);

例：

algname(list.begin(), list.end(), [](const string &value){

return value == "cd";

});

If版本

有些算法有支持谓词操作（自定义操作），使用\*\_if版本而不是重载

find\_if(list.begin(), list.end(), [](const string &value){

return value == "cd";

});