面向对象程序语言C++标准库

周雅倩
zhouyaqian@fudan.edu.cn
2016/4/1

主要内容

- Hello World程序
- 使用字符串
- 使用批量数据
- 组织程序和数据
- 使用序列式容器
- 使用库算法
- 使用关联式容器

使用库算法

- 很多容器操作可以用于多种容器
  - vector, string和list都允许调用insert函数来插入元素，也允许调用erase函数来删除元素。
  - 这些操作对于所有支持它们的类型都有同样的接口。
  - 每个容器都提供了相应的迭代器类型。

使用库算法

- 标准库通过相同的接口来提供很多标准算法。
- 通过使用这些算法，我们可以避免重复编写相同的代码。
- 程序会比用其他方式编写出来的代码会更小而且更简单
- #include<algorithm>

分析字符串

- 图案一般总是按行来组织的，这样就可以使用一个vector<string>来表示一幅图案，一行就是一个元素。
- 纵向连接两个图案：只要简单地把表示这两个图案的vector连接起来就可以：

```
vector<string> vcat(const vector<string> &top, const
    vector<string> &bottom)
{
    vector<string> ret=top;
    for(std::vector<string>::const_iterator it=bottom.begin();
    it!=bottom.end(); it++)
      ret.push_back(*it);
    return ret;
}
```


分析字符串

- 通过循环实现：

```
for(std::vector<string>::const_iterator it=bottom.begin();
      it!=bottom.end(); it++)
      ret.push_back(*it);
```

- 通过vector的insert函数实现：

```
ret.insert(ret.end(), bottom.begin(), bottom.end());
```

- 更为通用的解决方案：把复制元素和在容器末尾插入元素的概念分离开来。

```
copy(bottom.begin(), bottom.end(),back_inserter(ret));
```

泛型算法　　迭代器适配器

泛型算法

- copy(bottom.begin(), bottom.end(),back_inserter(ret));
- 泛型算法是一种不属于任何特殊容器的算法，它可以从其参数类型知道如何访问它使用的数据。
- 标准库的泛型算法常常在参数中带有迭代器，算法可以使用这些迭代器来操作底层容器的元素

copy算法

- copy(begin, end, out);
- 把区间[begin，end）中所有元素完全复制到out开始的元素序列中。
- 思考：
- copy(bottom.begin(), bottom.end(),ret.end())对吗?

分割字符串

- 任务：把一行英文字符串分成单个单词。
- 提示：这些单词通过空白符（空格，制表符，退格符或者是行终止符）来分隔。

- 思路: 通过计算i和j的值来对每个单词定位,每个单词都是区间[i, j)之间的字符

分割字符串—第五章方法

```
vector<string> split(const string & s)
{
    vector<string> ret;
    typedef string::size_type string_size;
    string_size i=0;

    while(i!=s.size()){
        while(i != s.size() && isspace(s[i])) i++;
        string_size j=i;
        for(; j != s.size() && !isspace(s[j]); j++);
        if(i!=j){
            ret.push_back(s.substr(i,j-i));
            i=j;
        }
    }
    return ret;
}
```


分割字符串的另一种方式

```
vector<string> split(const string & str){
    typedef string::const_iterator iter;
    vector<string> ret;
    iter i =str.begin();
    while(i!=str.end()){
        i=find_if(i,str.end(), not_space);
        iter j=find_if(i, str.end(), space);
        if(i!=str.end()){
            ret.push_back(string(i,j));
        }
        i=j;
    }
    return ret;
}

bool space(char c){
    return isspace(c);
}
bool not_space(char c){
    return !isspace(c);
}
```

i和j都是迭代器

复制找到的单词

find_if函数

```
i=find_if(i, str.end(), not_space);
```

迭代器　迭代器　谓词

```
j=find_if(i,  str.end(), space);
```

isspace函数

```
bool space(char c){
   return isspace(c);
}
```

说明：isspace是个重载的函数。适用于使用其他字符类型的语言，例如中文（wchar_t）。

问题：为何不直接使用isspace作为谓词？

回答：把一个重载的函数直接作为参数传给一个泛型函数是不可以的。

回文

查找URL

比较计算学生成绩的方案

- 两个方案：
  - 使用平均值，把学生没有提交的家庭作业成绩记作0分
  - 使用学生实际提交的家庭作业成绩的中值
- 需要对提交所有家庭作业的学生的中值成绩和漏交一次或多次的学生的中值成绩作比较。
- 解决两个子问题
  - 读取所有学生的纪录，把提交所有作业的学生和其他学生分割开。
  - 对每组中的所有学生分别使用上述两种方案，并报告每组的中值成绩。

检查一个学生是否完成所有作业

```
bool did_all_hw(const Student_info & s)
{
   //检测homework所有的值中是否包含0
   return((find(s.homework.begin(),s.homework.end(), 0))
                  ==s.homework.end());
}
```

如果find函数没有找到需要的值，就返回其第二个参数

处理学生纪录

```
vector<Student_info> did, didnt;
Student_info record;
while(read(cin, record)){
      if(did_all_hw(record))
            did.push_back(record);
      else
            didnt.push_back(record);
}
if(did.empty()){
      cout << "No student did all the homework!" << endl;
      return 1;
}
if(didnt.empty()){
      cout << "Every student did all the homework!" << endl;
      return 1;
}
```

检查容器是否为空

分析成绩

- 任务：对每组完成三个分析
  - 中值
  - 平均值
  - 已交作业的中值
- 解决方案：
  - 最简单：定义三个分析函数

解决方案

- 定义一个带5个参数的输出函数：
  - 输出流；
  - 表示分析名称的字符串；
  - 用于分析的函数；
  - 两个需要分析的vector对象。

```
write_analysis(cout,"median",median_analysis,did,didnt);
```


median_analysis函数

```
double median_analysis(const vector<Student_info> & students)
{
    vector<double> grades;
    transform(students.begin(),students.end(), back_inserter(grades), grade);
    return median(grades);
}

double grade_aux(const Student_info & s)
{
    try{
        return grade(s);
    }catch (domain_error){
        return grade(s.midterm, s.final, 0);
    }
}
```

重载函数???

指定需要转换的元素的区间

指定目的地

指定转换函数

```
transform(students.begin()
,students.end(),
back_inserter(grades),
grade_aux);
```


write_analysis函数

```
void write_analysis(ostream & out,  const string & name,
                    double analysis(const vector<Student_info>&),
                    const vector<Student_info> & did,
                    const vector<Student_info> & didn't)
{
    out << name << ": median(did) =" << analysis(did) <<
        ", median(didnt) = " << analysis(didnt) << endl;
}
```


计算vector的均值

```
double average(const vector<double> & v)
{
    return accumulate(v.begin(), v.end(),0.0)/v.size();
}
```

初值

<numeric>中定义

区间

累计所得的总和的类型和第三个参数的类型相同

基于家庭作业成绩的平均值来计算最终成绩

```
double average_grade(const Student_info & s)
{
    return grade(s.midterm, s.final, average(s.homework));
}
double average_analysis(const vector<Student_info> & students)
{
    vector<double> grades;
    transform(students.begin(),students.end(), back_inserter(grades),
    average_grade);
    return median(grades);
}
```

已提交的家庭作业成绩的中值

从homework中把非0元素提取出来

```
double optimistic_median(const Student_info & s)
{
    vector<double> nonzero;
    remove_copy(s.homework.begin(), s.homework.end(),
    back_inserter(nonzero), 0);

    if(nonzero.empty())
        return grade(s.midterm, s.final, 0);
    else
        return grade(s.midterm, s.final, median(nonzero));
}
```


把学生分类并且重新解决一个问题

- 问题：把不及格学生纪录添加到另外一个vector中，并且从已有的vector中删除这些学生的纪录
- 解决方案：
  - 使用list
  - 使用库算法

访问两次的解决方案

```
vector<Student_info> extract_fails(vector<Student_info> & students)
{
    vector<Student_info> fail;
    remove_copy_if(students.begin(),students.end(),back_inserter(fail),pgrade
    );
    students.erase(remove_if(students.begin(),students.end(),pgrade),
    students.end());
    return fail;
}
```

remove_copy_if(b,e,d,p):把[b, e)中所有使得谓词p为假的元素复制到d表示的目的地

remove_if(b,e,p):排列容器，使得[b, e)中的元素中，能使得谓词p为假的元素都位于这个区间的头部，返回一个指向这个区间中“不被删除”的元素之后的那个位置的迭代器

一次访问的解决方案

算法、容器和迭代器-1

- 算法是作用在容器元素上的
- ----它们并不作用在容器上。

remove_if(students.begin(),students.end(),pgrade)

- 这个调用并没有改变students的大小，只是能使得谓词p为假的元素都位于这个区间的头部，然后保持其他元素不变。当需要缩短这个vector时，我们必须自己完成：

students.erase(remove_if(students.begin(),students.end(),pgrade), students.end());

- 这里erase是vector的一个成员,它直接作用于容器,而不是容器的元素

算法、容器和迭代器-2

- 容器的操作会使被删除的元素的迭代器无效
  - 例如erase, insert
- 所以在使用这些操作的时候，必须小心地保存迭代器的值。
- 思考：对于vector和string对象来说，这些操作对迭代器的影响是什么？