简单 STL

konjacq

LifeScience@NJU

2024 年 10 月 7 日

大纲

如何调用一个函数

考虑函数原型

```
char f(int x,float y);
```

我们在调用时需要指定函数的名字f，还需要给出函数需要的参数int x和float y，最后，我们还要准备一个char 类型变量z 来存储f 的返回值。
这就是说，我们需要

```
int x=1;
float y=0.2;
char z=f(x,y);
```

或者更直接地，

```
char z=f(1,0.2);
```

小练习

根据以下函数原型，哪种调用方式是正确的?

```
void sort(int *__first,int *__last);
#define N ...
int n,a[N];
```

A. `sort(a,n);`　B. `sort(n,a);`　C. `sort(a,a+n);`　D. `a.sort(n);`

答案：C。

提示：`int *` 表示一个指针，而一个数组的名字代表数组的首地址，且地址和整形的加减一般还是地址。

如果我们需要排序一个数组

更具体地，假设我们有一个数组

`a[5]={3,4,2,0,1};`

现在我们希望对其从小到大进行排序。
我们可以写出代码

```
for (int i=0;i<n;++i)
    for (int j=0;j<n;++j)
        if (a[i]>a[j]) swap(a[i],a[j]);
```

如果我们需要排序一个数组

但是如果我们又有一个数组

```
b[7]={0,3,6,2,4,5,1};
```

也需要排序，我们该如何复用上面的代码？
把排序抽象为一个过程（简单理解为函数即可），每次告诉它需要给哪些数排序！

```
void my_sort(/* ... */);
```

排序函数

那么，如何告诉函数我们需要给哪些数排序呢？
最简单的，我们需要告诉函数这些数放在哪里，有多少个。
同时我们注意到：

$$\text{开始位置} + \text{数量} - 1 = \text{结束位置}$$

这意味着，如果我们告诉函数这些数放置的开始和结束位置（即存储区间的左右端点），且这些数连续排列的话，程序就能知道我们要给哪些数排序。

排序函数

这里，我们不使用结束位置，而使用结束位置的下一个位置作为参数。
换句话说，我们用一个左闭右开区间来描述数的存储区间。

```
void my_sort(int *l,int *r) {
    for (int *i=l;i<r;++i)
        for (int *j=l;j<r;++j)
            if ((*i)>(*j)) swap((*i),(*j));
}
```

STL 中的排序函数

每次用到排序都写这么长的代码，不好记又不好调，怎么办？
由于排序是一个非常常见的需求，所以 C++ 标准模板库也准备了一个差不多的函数来方便使用者。

```
inline void sort(_RandomAccessIterator __first,
                 _RandomAccessIterator __last,
                 _Compare __comp);
```

或者，可以将这个函数原型简单视为

```
void sort(int *__first,int *__last,/* ... */);
```

STL 中的排序函数

```
void sort(int *__first,int *__last,/* ... */);
```

这个函数接受 3 个参数。第 1、2 个是待排序的数的存储空间左右端点(左闭右开)，第 3 个是排序的规则。不提供第 3 个参数时，默认地，排序规则为从小到大排序。
例如，有以下代码

```
int a[5]={3,4,2,0,1};
sort(a,a+5);
```

则其执行后，a 数组中依次为{0,1,2,3,4}。

小练习

以下代码执行后，b 数组中的结果是?

```
int b[7]={0,3,6,2,4,5,1};
sort(b,b+5);
```

答案：{0,2,3,4,6,5,1}。
提示：我们只排序了b 中第 0 个到第 $5-1=4$ 个元素，这些元素升序排列，剩余的元素不改变。

作业

1. 洛谷 P1177 【模板】排序。

除了排序以外

https://www.runoob.com/cplusplus/cpp-stl-tutorial.html

C++ 标准模板库（Standard Template Library，STL）是一套功能强大的 C++ 模板类和函数的集合，它提供了一系列通用的、可复用的算法和数据结构。
STL 的设计基于泛型编程，这意味着使用模板可以编写出独立于任何特定数据类型的代码。
STL 分为多个组件，包括容器（Containers）、迭代器（Iterators）、算法（Algorithms）、函数对象（Function Objects）和适配器（Adapters）等。

上面提到的`sort` 就是众多 STL 算法中的一种。

除了排序以外

除了用于排序的sort 以外，STL 还提供了

1. 用于翻转序列的reverse；
2. 用于随机打乱序列的random_shuffle；
3. 用于查找第 n 大元素的nth_element；
4. 用于二分查找的lower_bound 和upper_bound；
5. 用于生成排列的next_permutation 和prev_permutation；
6. 用于序列去重的unique。

接下来将依次介绍。

reverse

```
void reverse(int *__first,int *__last);
```

这个函数接受 2 个参数，依次是存储空间左右端点（左闭右开）。翻转后，最末尾的元素与第一个元素交换位置，次末尾的元素与第二个元素交换位置，依此类推。
这种左闭右开的参数传递方式我们还将在许多 STL 算法中见到。

random_shuffle

```
void random_shuffle(int *__first,int *__last);
```

这个函数接受 2 个参数，依次是存储空间左右端点（左闭右开）。算法将所有数随机排列。

nth_element

```
void nth_element(int *__first,int *__nth,int *__last);
```

这个函数接受 3 个参数，前后两个是存储空间左右端点（左闭右开），中间的是需要找的第 n 大的数的位置。算法将第 n 大的数放在数组中第 n 个位置上，与 n 相比较小的数均在 n 之前，较大的则在之后。
这个算法是 $O(n)$ 的。

小练习

根据以下函数原型，哪种调用方式是正确的?

```
void nth_element(int *__first,int *__nth,int *__last);
#define N ...
int n,m,a[N];
```

A. `nth_element(a,m,a+n);`

B. `nth_element(a,a+m,a+n);`

C. `nth_element(a,a+n);`

D. `nth_element(a,a+n,m);`

答案：B。

假设`a={3,4,2,0,1}`，且`n=4`、`m=2`，经过上述调用后，`a`数组中的结果是?

答案：`{0,2,3,4,1}`或`{2,0,3,4,1}`。

lower_bound 和 upper_bound

用于在有序序列上进行二分查找。
不过这个是讲到二分之后的事，所以今天不讲。

next_permutation 和 prev_permutation

用于生成排列。

```
bool next_permutation(int *__first,int *__last);
```

这个函数接受 2 个参数，依次是存储空间左右端点（左闭右开）。算法生成区间内数的下一个排列并返回`true`；如果当前排列已是最后一个，则生成第一个排列并返回`false`。

`prev_permutation` 类似理解即可。

其中，一个 1 到 n 的排列是一个长度为 n 的整数数列，其中每个数均在 $[1,n]$ 范围内，且没有两个数相同。两个排列之间的顺序由字典序决定，也即对于两个 1 到 n 的排列

$$p_1 < p_2 \iff \exists i \in [i,n] \text{s.t.} \begin{cases} p_{1,j} = p_{1,j} & ,j < i \\ p_{1,j} < p_{1,j} & ,j = i \end{cases}$$

容易证明这是一个全序关系。

unique

```
int *unique(int *__first,int *__last,/* ... */);
```

这个函数接受 3 个参数。第 1、2 个是待去重的数的存储空间左右端点(左闭右开)，第 3 个是判断相等的方法。不提供第 3 个参数时，默认地，使用==。
同时，这个函数返回指向去重后最后一个函数的下一个位置的指针，所以你可能想写

```
int n,a[N];
n=unique(a,a+n)-a;
```

即，将n 设置为去重后a 中元素个数。注意这个函数并不真的去掉重复元素，而只是把第 2 或更多次出现的元素移动到序列最后。

什么是容器

一般地，我们用容器来装东西。
一般地，我们用数据结构来保存数据。
这两个看起来就很像，所以目前为止，我们可以将容器简单地理解为封装的数据结构。也即，容器以特定方式保存一些数据。
注意，在这里我们不确切地区分容器和适配器。适配器可以将一种容器包装为另一种容器；由于包装前后的容器使用方式类似，因此我们暂时将适配器也视作一种容器。

吃午饭

假设我们立刻下课，然后去吃午饭。
那么，我们应当在食堂打饭的地方排成队列，先到的先打，后到的后打。
计算机中，“队列”则是这样一种数据结构，它

1. 保存一些数，可以放入和取出；
2. 先放入的数先被取出，后放入的数后被取出。

STL 当中提供了queue 作为队列，包含在<queue> 头文件中。
queue 的原型为

```
template <class T,class Container=deque<T>> class queue;
```

更一般地，一个保存int 类型的队列q 通常通过

```
queue<int> q;
```

来定义。当然你也可以换成别的你需要的类型和需要的变量名。

队列

对于队列q，STL 提供的、常用的成员函数有

```
q.empty(); /* 返回q是否为空 */
q.size();  /* 返回q中的元素数量 */
q.front(); /* 返回最早放入的元素 */
q.back();  /* 返回最晚放入的元素 */
q.push(x); /* 将x放入队列 */
q.pop();   /* 取出最早放入的元素 */
```

小练习

以下代码的输出是?

```
queue<int> q;
int x=0;
q.push(1);
printf("%d ",x);
q.push(2);
printf("%d ",x);
x=q.front();
printf("%d ",x);
q.push(3);
printf("%d ",x);
x=q.back();
printf("%d ",x);
q.pop();
printf("%d ",x);
x=q.front();
printf("%d ",x);
x=q.back();
printf("%d ",x);
if (q.empty())
    printf("YES ");
else printf("NO ");
q.pop();
if (q.empty())
    printf("YES ");
else printf("NO ");
q.pop();
if (q.empty())
    printf("YES ");
else printf("NO ");
```

答案: 0 0 1 1 3 3 2 3 NO NO YES。

书

考虑一些书从下到上累成一摞。
那么，下面的书一定被先放上去，上面的书被后放上去；而拿的时候，则应该从上面的书开始拿。
计算机中，“栈”则是这样一种数据结构，它

1. 保存一些数，可以放入和取出；
2. 先放入的数后被取出，后放入的数先被取出。

栈看起来就和队列很像，所以其定义和使用方式也与队列几乎一致。STL 当中提供了`stack` 作为队列，包含在`<stack>` 头文件中。
一个保存`int` 类型的队列`q` 通常通过

```
stack<int> s;
```

来定义。

栈

对于栈s，STL 提供的、常用的成员函数有

```
s.empty(); /* 返回s是否为空 */
s.size();  /* 返回s中的元素数量 */
s.top();   /* 返回最晚放入的元素 */
s.push(x); /* 将x放入队列 */
s.pop();   /* 取出最晚放入的元素 */
```

注意：在 STL 提供的栈中我们没法访问最早放入的元素了。然而，自己写的栈或者别的 STL 容器可以帮助我们完成这一功能，将在后续过程中讲到。

小练习

以下代码的输出是?

```
stack<int> s;
int x=0;
s.push(1);
printf("%d ",x);
s.push(2);
printf("%d ",x);
x=s.top();
printf("%d ",x);
s.push(3);
printf("%d ",x);
x=s.top();
printf("%d ",x);
s.pop();
printf("%d ",x);
x=s.top();
printf("%d ",x);
if (s.empty())
    printf("YES ");
else printf("NO ");
s.pop();
if (s.empty())
    printf("YES ");
else printf("NO ");
s.pop();
if (s.empty())
    printf("YES ");
else printf("NO ");
```

答案: 0 0 2 2 3 3 2 NO NO YES。

双端队列

计算机中，“双端队列”是这样一种数据结构，它

1. 保存一些数，可以放入和取出；
2. 放入时，可以从队列前面或后面放入；
3. 取出时，可以从队列前面或后面取出。

STL 当中提供了`deque` 作为双端队列，包含在`<queue>` 头文件中。一个保存`int` 类型的双端队列`d` 通常通过

```
deque<int> d;
```

来定义。

双端队列

对于双端队列d，STL 提供的、常用的成员函数有

```
d.empty();          /* 返回d是否为空 */
d.size();           /* 返回d中的元素数量 */
d.front();          /* 返回队首的元素 */
d.back();           /* 返回队尾的元素 */
d.push_front(x);    /* 将x放至队首 */
d.push_back(x);     /* 将x放至队尾 */
d.pop_front();      /* 取出队首的元素 */
d.pop_back();      /* 取出队尾的元素 */
```

向量

考虑一种数据结构，支持

1. 按顺序保存一些数；
2. 访问和修改任意位置上的数（到目前为止听起来像数组）；
3. 增加和减少保存数的个数。

上述数据结构即是 STL 当中的向量`vector`，包含在`<vector>` 头文件中。一个保存`int` 类型的向量`v` 通常通过

```
vector<int> v;
```

来定义。也可以通过

```
vector<int> v(n);
vector<int> v(n,m);
```

来创建一个含`n` 个元素（且元素默认值为`m`）的`vector`。

向量

对于向量v，STL 提供的、常用的成员函数（和方法）有

```
v[x];              /* 访问v的第x个元素 */
v.empty();         /* 返回v是否为空 */
v.size();          /* 返回v中的元素数量 */
v.resize(x);       /* 将v中的元素数量修改为x，延长时可指定第二
    个参数以提供默认值，
    缩短时会丢弃后面的元素 */
v.clear();         /* 清空v中所有元素 */
v.push_back(x);    /* 将x加入到v的最后一个元素之后 */
v.pop_back();      /* 移除v的最后一个元素 */
v.begin();         /* 返回v的第一个元素的地址 */
v.end();           /* 返回v的最后一个元素之后的地址 */
```

这就意味着，我们可以使用

```
sort(v.begin(),v.end());
```

来对向量v 进行排序。

优先队列

计算机中，“优先队列”是这样一种数据结构，它

1. 保存一些数，可以放入和取出；
2. 取出时，最大的数先被取出。

更多时候，我们也将其称之为“堆”。其中，优先取出最大数的叫“大根堆”，反之则叫“小根堆”。在下面，我们默认（当然，也是 STL 默认）提到的优先队列和堆都是大根堆。STL 当中提供了`priority_queue` 作为优先队列，包含在`<queue>` 头文件中。
一个保存`int` 类型的优先队列`p` 通常通过

```
priority_queue<int> p;
```

来定义。

优先队列

对于优先队列p，STL 提供的、常用的成员函数有

```
p.empty(); /* 返回p是否为空 */
p.size();  /* 返回p中的元素数量 */
p.top();   /* 返回最大的元素 */
p.push(x); /* 将x放入队列 */
p.pop();   /* 取出最大的元素 */
```

小练习

如何得到一个小根堆?
提示:

1. 观察优先队列的原型

```
template <class T,class Container=vector<T>,
    class Compare=less<typename Container::value_type>>
    class priority_queue;
```

可以发现其默认比较函数为`less<int>`。通过将其换为`greater<int>`，即通过

```
priority_queue<int,vector<int>,greater<int>> p;
```

可定义小根堆。

2. 放入和取出时分别取负数，即可使大小关系反转。

使用`sort` 对数组排序时也可通过在排序后调用`reverse` 来实现降序排列。

扩展：比较器

真的要讲这个吗？

作业

1. 洛谷 B3614 【模板】栈
2. 洛谷 B3616 【模板】队列
3. 洛谷 B3656 【模板】双端队列 1
4. 洛谷 P3378 【模板】堆

额外地，可以尝试

1. 洛谷 B3666 求数列所有后缀最大值的位置
2. 洛谷 P2032 扫描

Q&A

1. QQ: 732634033；
2. Email: konajcq@outlook.com。

当然也可以选择直接找 IAD。