[15] 线性表

深入浅出程序设计竞赛

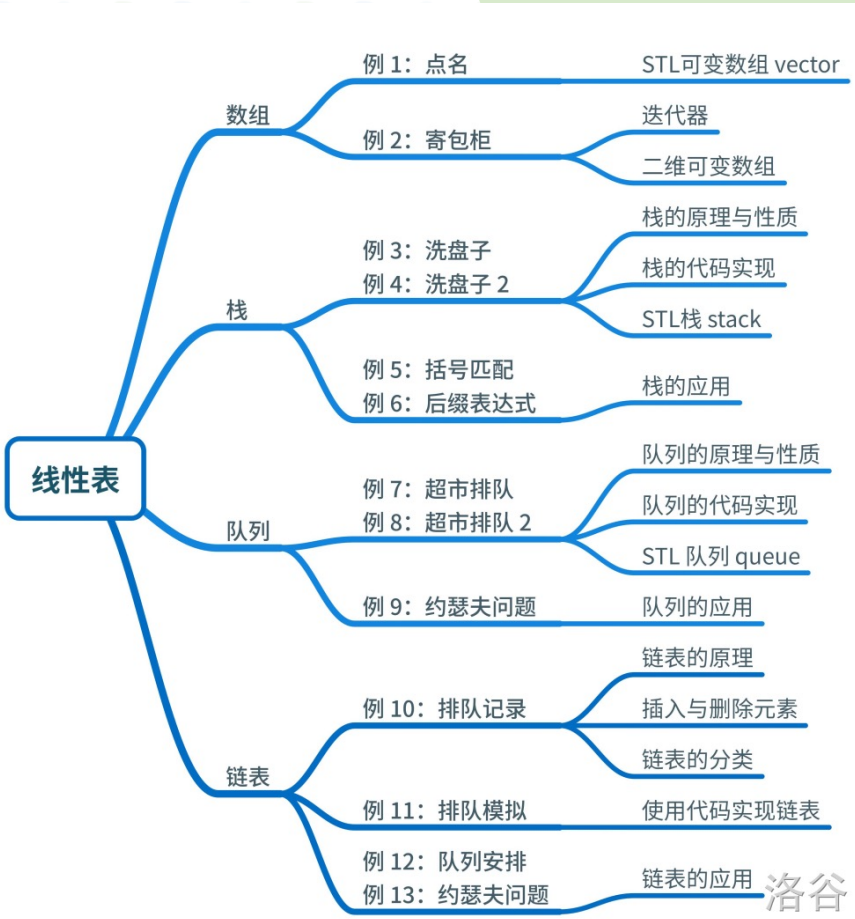
第 3 部分 – 简单数据结构

V 2021-02

版权声明

- 其它《深基》配套资源、购买本书等请参阅：https://www.luogu.com.cn/blog/kkksc03/IPC-resources
- 如果课件有任何错误，请在这里反馈 https://www.luogu.com.cn/discuss/show/296741

本章知识导图


线性表
数组
例 1：点名
STL可变数组 vector
例 2：寄包柜
迭代器
二维可变数组
栈
例 3：洗盘子
例 4：洗盘子 2
栈的原理与性质
栈的代码实现
STL栈 stack
例 5：括号匹配
例 6：后缀表达式
栈的应用
队列
例 7：超市排队
例 8：超市排队 2
队列的原理与性质
队列的代码实现
STL 队列 queue
例 9：约瑟夫问题
队列的应用
链表
例 10：排队记录
链表的原理
插入与删除元素
链表的分类
例 11：排队模拟
使用代码实现链表
例 12：队列安排
例 13：约瑟夫问题
链表的应用

第 15 章 线性表

数组

本书已经在第一部分介绍了数组的定义和使用，但是在这里可以重新审视一下作为数据结构的数组。

请翻至课本 P204

询问学号

例 15.1 （洛谷 P3156）

有 n（$n \leqslant 2\times10^6$） 名同学进入教室。每名同学的学号在 1 到 10^9 之间，按进教室的顺序给出。

老师想知道第 i 个进入教室的同学的学号是什么？最先进入教室的同学 i=1，询问次数不超过 10^5 次。

```
10 3
1 9 2 60 8 17 11 4 5 14
1 5 9
```

```
1
8
5
```

询问学号

建立数组，按照顺序来记录按顺序到达的同学的学号，直接在数组中查询对应下标的数据。

建立数组的大小要超过最多可能的同学总数量，也就是 2×10^6。

```
int n,m,a[2000050];

for (int i = 1; i <= n; i++)
    cin >> a[i];
while (m--) {
    int i;
    cin >> i;
    cout << a[i] << endl;
}
```

可变长度数组

但是有的时候数组要开多大比较难以计算，无法确定。

在 C++ 的 STL 中，给我们提供了一个可变长度数组。可变长度数组的头文件是 <vector>。vector 有几个常用功能：

功能	说明
vector<int> v(N,i)	建立一个可变长度的 int 数组 v，且初始有 N 个为 i 的元素。N,i 可以省略。
v.push_back(a)	将元素 a 插入 v 的末尾。
v.size()	返回元素个数。
v.resize(n,m)	重新调整数组大小为 n。如果 n 比原来大，则新增的部分都初始化为 m。
v[a]	访问下标为 a 的元素。

可变长度数组

尝试用 vector 解决这道题目。读入每个学生的学号，并且使用 push_back 把读入的学号塞入 vector 中。

```
vector<int> stu;
cin >> n >> m;
for(int i = 0; i < n; i++) {
    cin >> tmp;
    stu.push_back(tmp);
}
```

使用类似数组的索引，在 stu 中获取第 i 个进入教室的同学。注意，vector 的下标也是从 0 开始的。

```
for(int i = 0; i < m; i++) {
    cin >> tmp;
    cout << stu[tmp - 1] << endl;
}
```

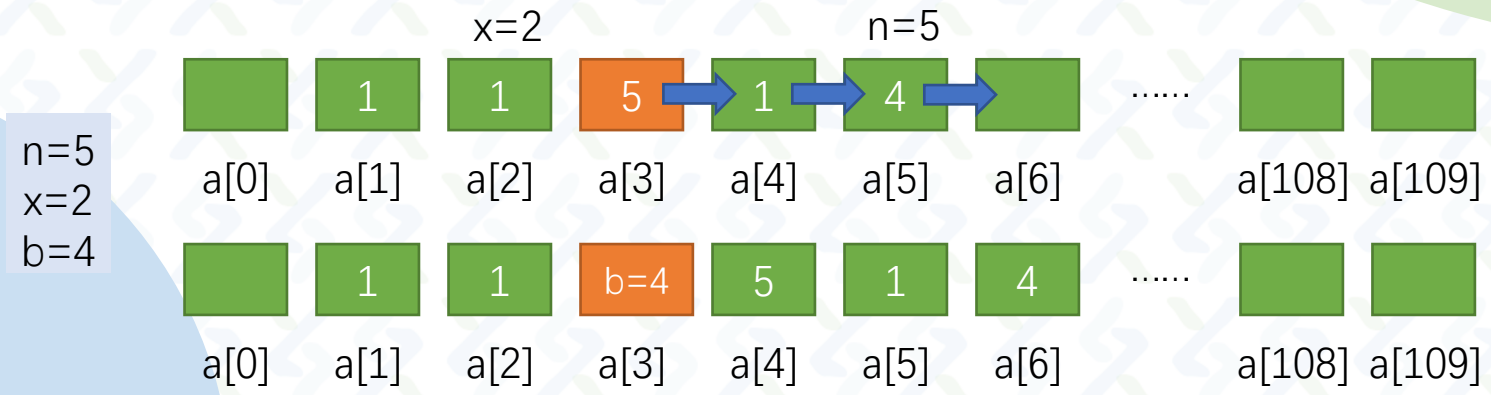
询问学号

注意：使用方括号索引来访问数组元素时，数组的大小必须不小于索引，否则就和访问普通数组越界一样而访问无效内存。

可通过数组初始化、push_back 或 resize 成员函数来增加数组长度。

数组的特点和局限性：

- 存储查询给定索引（下标）的数据：效率很高，复杂度 O(1)
- 将整个数组的一段数据进行插入或删除操作，或者搜索指定元素（如果没有排序）：效率很低，时间复杂度 O(n)。



栈

古代的货栈只有一个门用于运入或运出货物。当货栈堆满时，为了从中运出货物，往往是最近放的货物最先拿出，以前放的货物会后拿出。

请翻至课本 P207

洗盘子

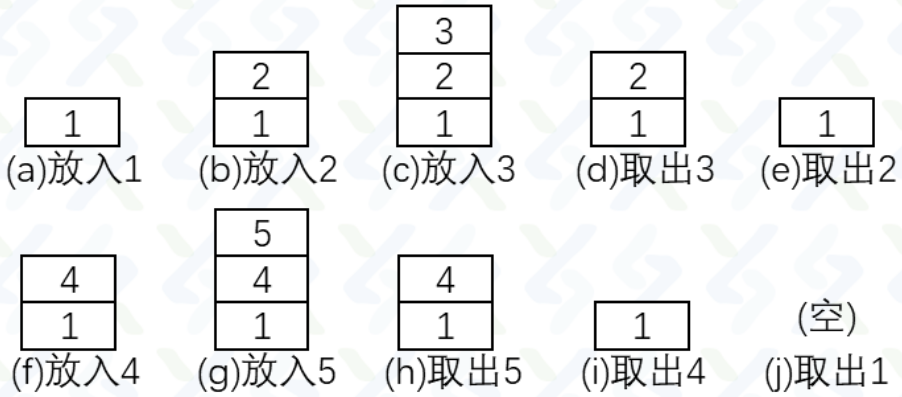
例 15.3

想象一下打工时的洗盘子过程。最开始桌子上没有待洗餐盘。

客人吃完饭，依次把 1、2、3 这三个盘子放在了桌子上。

小止取出最顶端的 3 号盘子洗，洗完后取出现在的顶端盘子 2 号。

又有一个客人吃完了饭，依次放进了 4、5 这两个盘子。在这之后，小止依次洗完了 5、4、1 这三个盘子，结束了工作。



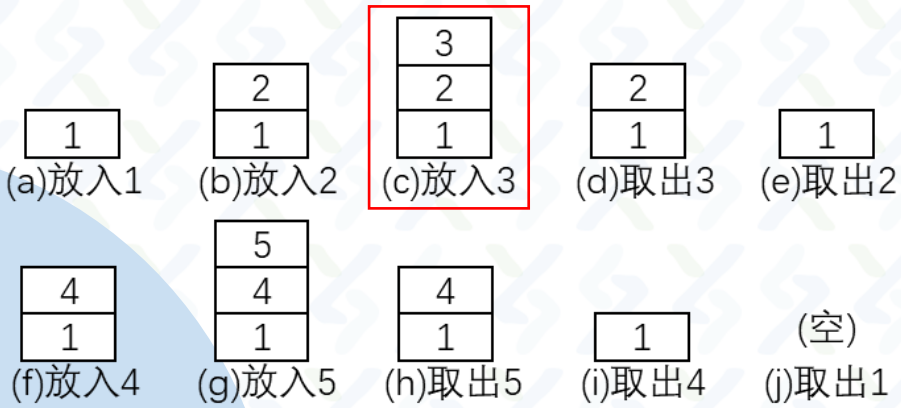
洗盘子

每次只会取出最上面的盘子，而最上面的盘子又是最新放入盘子堆的。越下面（也就是越早放入这一堆）的盘子，则越晚被取出。

也就是说，对于这叠餐盘，如果a比b早加入，那一定a比b后退出。

能够满足这个操作的数据结构，就被称为栈。

栈是一种“后进先出”的线性表，其限制是仅允许在表的一端进行插入和删除运算，这一端被称为栈顶。



初始空栈：p=0

放入3后：p=3

栈

例 15.4

编写程序模拟小止洗完的过程。一个栈可以提供下面几个功能：

void push(x)：将x压入栈。对应例子中的“放餐盘”。

void pop()：弹出栈顶的元素。对应“取走餐盘”。

int top()： 查询栈顶的元素由此知道接下来该洗哪个餐盘。

栈

记录一个栈顶指针 p，指向下一个待插入的数组位置。

push 操作：直接把 x 赋值给 stack[p]，并且更新指针加一即可。

```
void push(int x){
    stack[p] = x; p += 1;
}
```

pop 操作：直接让栈顶指针后退一位。

```
void pop(){
    p -= 1;
}
```

top 操作：因为 p 指向下一个待插入的数组位置，所以栈顶的数组位置是 p-1。

```
int top(){
    return stack[p - 1];
}
```

栈

C++ STL 提供了栈。它的头文件是 <stack>，有以下的方法：

功能	说明
stack <int> s	建立一个栈 s，其内部元素类型是 int。
s.push(a)	将元素 a 压进栈 s。
s.pop()	将 s 的栈顶元素弹出。
s.top()	查询 s 的栈顶元素。
s.empty()	查询 s 是否为空。
s.size()	查询 s 的元素个数。

栈

在使用栈的时候，需要防止栈因存储内容过多而导致溢出，也需要防止对空栈弹出元素。

C++ STL 已经做了一部分处理，但是查询空栈的栈头也是会导致 Runtime Error 的。

对于手写栈可以在操作之前进行判断。例如弹出操作：

```
void pop(){
    if (p == 0) printf("Stack is empty");
    else p -= 1;
}
```

注意：STL 虽然提供了许多方便的功能，但是如果使用 STL 时不打开 -O2 优化开关，就有一点慢（常数大）。在需要追求运行速度的情况下，往往需要自己手写栈。

括号匹配

例15.5 （Uva 673）

给定若干个字符串，每个字符串由()[]{}这六个字符构成。如果所有的括弧都可以匹配，例如[([])[]]，那么这个字符串合法，否则非法。

试判断一个字符串是否合法。

```
3
([])
(([()]))
([()[]()])()
```

```
Yes
No
Yes
```

括号匹配

思路：我们假设有一个字符串[([]){}]。对于每个右括号去找匹配的左括号，匹配则删去，类似消消乐。

```
STEP  1: [
STEP  2: [(
STEP  3: [([
STEP  4: [([] ← 发现 [] 并消去
STEP  5: [(
STEP  6: [() ← 发现 () 并消去
```

```
STEP  7: [{
STEP  8: [{} ← 发现 {} 并消去
STEP  9: [
STEP 10: [] ← 发现 [] 并消去
STEP 11: 读入完毕，已全部消去
```

如果处理字符串的所有字符，发现还有剩下括号，那么就说明有些括号没有被匹配到，说明是非法的括号序列。比如([)(])。

括号匹配

首先编写一个 trans 函数进行括号匹配。

```
char trans(char a){
    if (a == ')')return '(';
    if (a == ']')return '[';
    if (a == '}')return '{';
    return '\0';
}
```

当括号不匹配的时候，栈内可能还存有元素，这会对之后判断括号匹配产生干扰。

因此在匹配之前需要清空栈。

```
while(!s.empty()) s.pop();
```

括号匹配

接下来依次对字符串用栈处理即可完成本题。

如果栈为空，直接放入栈中；

如果发现读到的字符和栈顶的字符可以匹配，那么就消去。

```
getline(cin, p); //读入一行
for (int i=0; i < p.size(); ++i) {
    if (s.empty()) {
        //如果栈为空，直接放入栈中
        s.push(p[i]);
        continue;
    }
    if (trans(p[i]) == s.top())
        s.pop();
    else s.push(p[i]);
}
if (s.empty())printf("Yes\n");
else printf("No\n");
```

C++ 读入字符串

C++ 中处理字符串问题的一个注意点：

使用 cin 读入一个独占的数字后，其读入指针在这一行的末尾。

使用 getline 读入一行字符串时，只会读到空串（第一行）。如果希望读到第二行，则必须要假装读入这一行，可以使用 getline，也可以使用 getchar 等。

思考：如果没有第三行的 getline 会有什么后果？

```
cin >> num;
string p;
getline(cin, p); // 假装换行符，读入空行
while (num--) {
    getline(cin, p);
    //读入一行，这样这个 p 是正常的。
    ……
```

```
3
([])
(([()]))
([()[]()])()
```

后缀表达式

例15.6（洛谷 P1449）

后缀表达式是不再引用括号，运算符号放在两个运算对象之后，所有计算按运算符号出现的顺序，严格地由左而右新进行的表达式，且不必考虑运算符优先级。

例如输入2.4.*1.3.+-@，其中.是每个数字的结束标志，@是整个表达式的结束标志。输出4。

```
STEP  1: 2
STEP  2: 2 4
STEP  3: 2 4 *
STEP  4: 8 ← 发现 * 并计算
STEP  5: 8 1
STEP  6: 8 1 3 +
```

```
STEP  7: 8 4 ← 发现 + 并计算
STEP  8: 8 4 -
STEP  9: 4
```

后缀表达式

每次读到一个运算符，就取出在它前面的次近和最近的数字进行相应运算后再把它放入原来的序列。

也就是每次操作，不是获得并且弹出序列的一端的前两个数字，就是往这一端放入一个数字，符合栈的功能，用栈完成这个问题。

```
if (ch == '.')
    n.push(s), s = 0;
else if (ch != '@') {
    x = n.top(); n.pop(); y = n.top(); n.pop();
    switch (ch) {
        case '+': n.push(x + y); break;
        case '-': n.push(y - x); break;
        case '*': n.push(x * y); break;
        case '/': n.push(y / x); break;
    }
}
```

队列

在超市种，收银员会给排在队伍最前面的顾客买单，然后服务队伍中下一个顾客。而队伍的末尾也一直会有更多的顾客依次加入队列。

请翻至课本 P211

超市排队

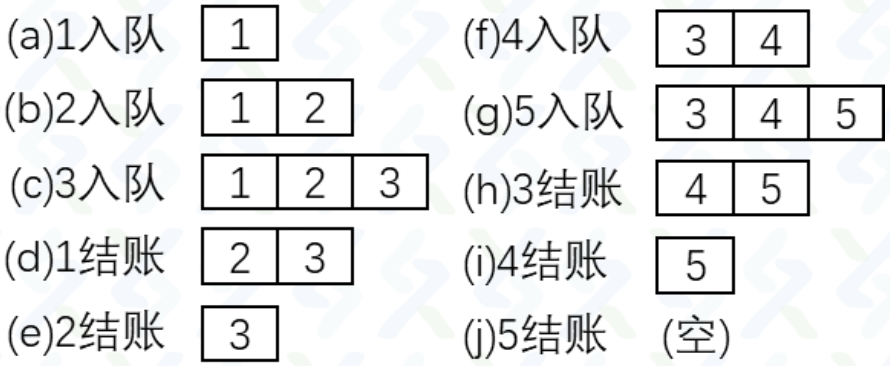
例 15.3

想象一下超市购物，最开始时，收银台前面一个人都没有。

然后 1、2、3 号顾客排入了队列。收银员帮 1 结账后又帮 2 结账。

4、5 号顾客又加入了队列。收银员又帮 3、4、5 号顾客结账。

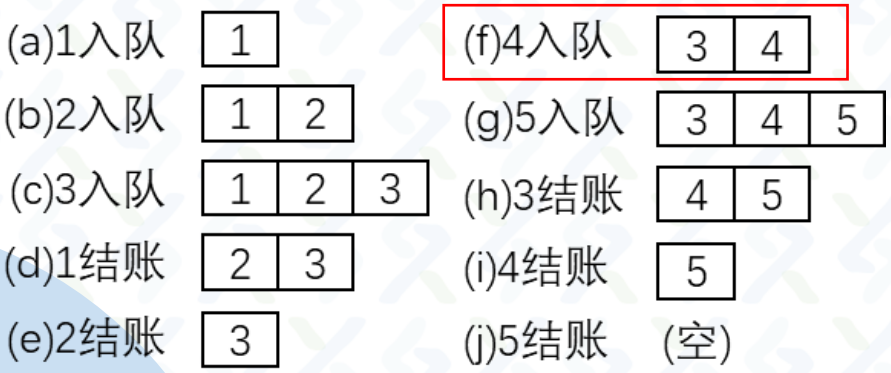
此时所有顾客都已经结账，收银台前又没有人了。

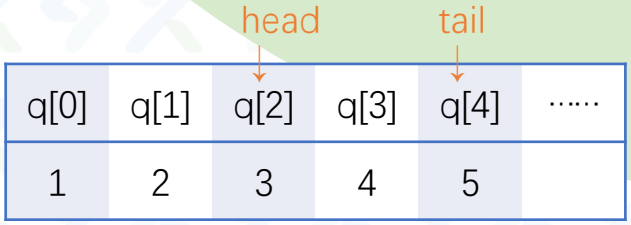


超市排队

先排队的人先结账完毕。这种数据结构就是队列。它的特点是先进先出。可以在队列的一端插入元素，在另一端删除元素。

在存储逻辑上，可以认为顾客是站着不动的，而有两个指针——头指针（head）和尾指针（tail）来控制队列里的元素。





初始空队列：head=tail=0

队列为[3,4]：head=2, tail=4

队列

例 15.4

请写程序模拟收银员的过程，也就是完成三个函数：

void push(x)：将x压入队列，即一个人来排队，站在队尾。

void pop()：弹出队首的元素，即最前面的人结完账，离开队列。

int front()：查询队首的元素，即给谁结账。

队列

队列是有头（head）尾（tail）的。需要记录队列的头和尾在哪里。

push 操作：直接把 x 赋给队尾，再让队尾 +1。

```
void push(int x){
    queue[tail] = x; tail += 1;
}
```

pop 操作：不必让元素全部往前进一位，只需让头指针前进。

```
void pop(){。
    head += 1;
}
```

front 操作：直接获取 q[head] 即可。

```
int front(){
    return queue[head];
}
```

队列

C++ STL 提供了队列。它的头文件是 <queue>，有以下的方法：

功能	说明
queue <int> q	建立一个队列 q，其内部元素类型是 int。
q.push(a)	将元素 a 插入队列 q 的末尾。
q.pop()	将 q 的队首元素删除。
q.front()	查询 q 的队首元素。
q.end()	查询 q 的队尾元素。
q.size()	查询 q 的元素个数。
q.empty()	查询 q 是否为空。

约瑟夫问题

例15.9 （洛谷 P1996）

n 个人围成一圈，从第一个人开始报数,数到 k 的人出列，再由下一个人重新从 1 开始报数，数到 k 的人再出圈，依次类推，直到所有的人都出圈。

请输出依次出圈人的编号。1≤n,k≤100。

10 3	3 6 9 2 7 1 8 5 10 4

约瑟夫问题

思路：

问题可以转化为这 n 个小朋友在队列中，每次操作使在队首的人跑到队尾。每 k 次操作删去队首。

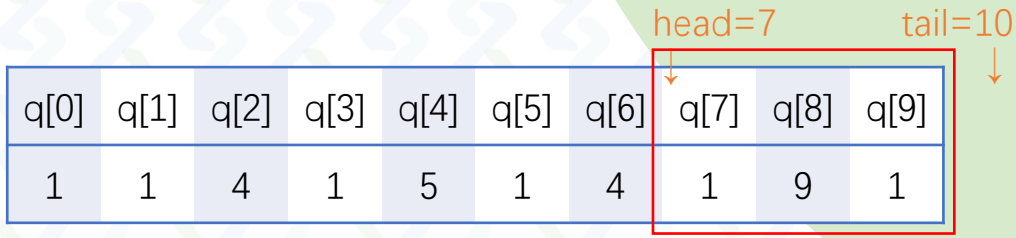
用队列模拟这个过程。

```
STEP 1:1 2 3 4 5 6 7 8 9 10
STEP 2:2 3 4 5 6 7 8 9 10 1
STEP 3:3 4 5 6 7 8 9 10 1 2
删去队首
STEP 4:4 5 6 7 8 9 10 1 2
STEP 5:5 6 7 8 9 10 1 2 4
STEP 6:6 7 8 9 10 1 2 4 5
删去队首
STEP 7:7 8 9 10 1 2 4 5
以此类推。
```

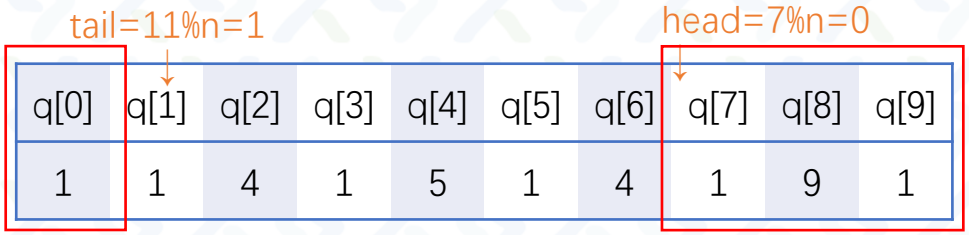
```
for(i=1;i<=n;i++) q.push(i);
while(q.size() != 1) {
    for(i=1;i<k;i++) {
        q.push(q.front());
        q.pop();
    }
    printf("%d\n",q.front());
    q.pop();
}
```

循环队列

假设定义 queue[10] 的队列，进进出出，头尾指针可能超过 100 导致假溢出。但实际上队列不长，造成数组空间浪费。



可使用循环队列的方式映射指针，增加空间利用率，队内长度不超过 n 即可。



如果使用 STL 队列，那么就可以不用考虑循环队列了，方便但慢。

链表

一排人手牵手站在一起，其中有一个人离队，剩下人保持原来顺序牵着手使队伍相连。那么每个人所牵着手的人有没有变化呢？

请翻至课本 P214

排队记录

例 15.10

n 名同学在排队进入洛谷大厦。每个人都只知道自己后面是谁，且编号为 1 的人站在最前面。能否还原出这个队伍？

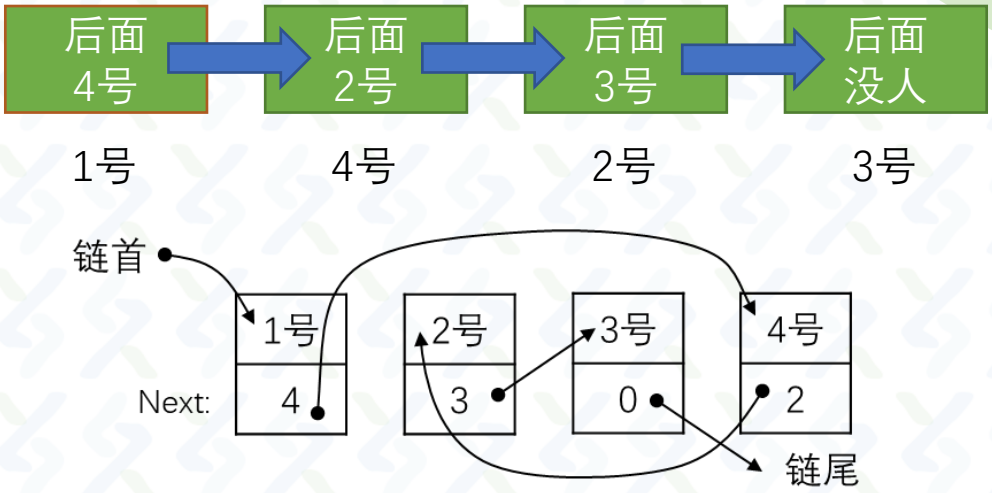
提示：只要知道第一个人后面是谁，就知道他的后面的后面是谁，那么就知道他后面的后面的后面……

排队记录

知道每人前/后面是谁，从而得到整个序列的数据结构就是链表了。

```
Next[1]=4; Next[2]=3; Next[3]=0; Next[4]=2;
for(int i=1; i != 0; i = Next[i])
    printf("%d ",i);
```

假设这个队伍为 1 4 2 3 ：

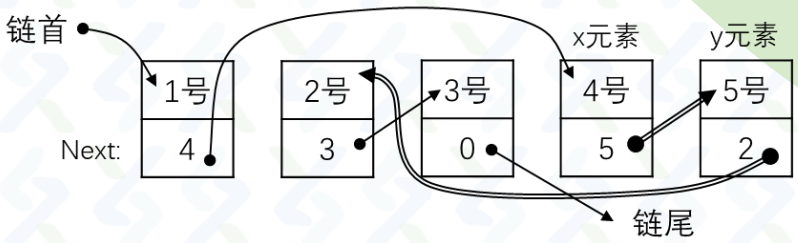


链表

此时 5 号同学插队到 4 号后面，那么会发生什么？

5 号同学的后面的人是 4 号同学原本后面的人，也就是 2 号；而 4 号同学的后面的人变成了插队进来的 5 号。

这是链表增加一个节点的操作。



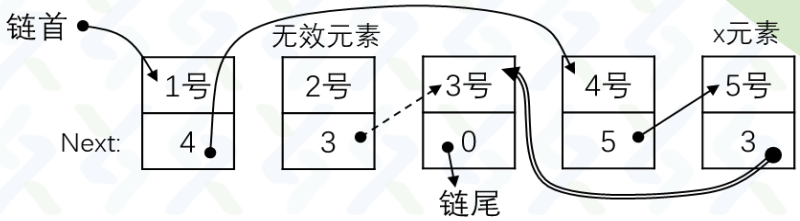

```
void insert(int x,int y) {
    Next[y]=Next[x];
    Next[x]=y;
}
```

链表

此时 2 号同学离开了队伍，那么会发生什么？

仅仅是 5 号同学的后面一位同学不再是 2 号，而是 2 号同学的后面一位同学，也就是 3 号。

这是链表删除一个节点的操作。




```
void remove(int x) {
    Next[x]=Next[Next[x]];
}
```

链表

例 15.11

实现一个数据结构，维护一张表（最初只有一个元素 1）。需要支持下面的操作，其中 x 和 y 都是 int 范围内的正整数，且都不一样，操作数量不多于 2000：

ins_back(x,y) ：将元素 y 插入到 x 后面；

ins_front(x,y) ：将元素 y 插入到 x 前面；

ask_back(x) ：询问 x 后面的元素；

ask_front(x) ：询问 x 前面的元素；

del(x)：从表中删除元素 x，不改变其他元素的先后顺序。

链表

思路：使用双向链表。首先每次操作之前，都需要知道一个元素在表中的编号。

```
int find(int x) {
    int now = 1;
    while (now && s[now].key != x)now = s[now].nxt;
    return now;
}
```

ins_back：注意更新后继的前驱以及前驱的后继，这里 tot 代表使用了多少节点的位置。ins_front 同理。

```
void ins_back(int x, int y) {
    int now = find(x);
    s[++tot] = node(y, now, s[now].nxt);
    // 分别为元素、前驱、后继
    s[s[now].nxt].pre = tot;
    s[now].nxt = tot;
}
```

链表

ask_back：只需根据编号获得其后继的值即可。ask_front 同理。

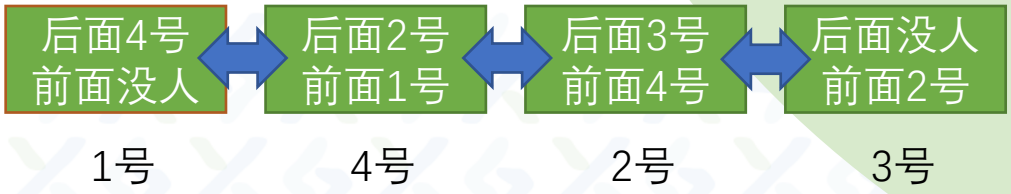
```
int ask_back(int x) {
    int now = find(x);
    return s[s[now].nxt].key;
}
```

del：删除一个元素时，只需让这个元素的前驱的后继变成它的后继，它的后继的前驱变成它的前驱即可。

```
void del(int x) {
    int now = find(x);
    int le = s[now].pre, rt = s[now].nxt;
    s[le].nxt = rt;
    s[rt].pre = le;
}
```

双向链表

有时候只知每个元素后面的元素是不够的，还需要知道前面的元素是什么。可以使用 双向链表，同时记录每个节点的前驱和后继，然后就可以往两个方向遍历啦。



实现双向链表，可使用多个数组，也可使用结构体。具体的创建、查找、增加、删除的代码和说明请参阅课本 P216。

```
struct node {
    int pre, nxt;        // 分别记录前驱和后继结点在数组s中的下标
    int key;             // 结点的值
    node(int _key = 0, int _pre = 0, int _nxt = 0) //结构体初始化
    {pre = _pre; nxt = _nxt; key = _key; }
};
node s[1005]; // 一个池。以后想要新建结点，就从s数组里面分配给新结点。
```

更多链表

链表有很多种，其中比较基础的如下：

种类	特点
单向链表	只记录每个节点的后继。
双向链表	记录每个节点的前驱和后继。
循环单向链表	单链表，最后一个节点后继为第一个节点。
循环双向链表	双链表，形成环形。

链表插入和删除的复杂度是 O(1)。

链表搜索指定元素位置/定位第k个元素的复杂度是 O(n)。

相比于数组，链表插入删除快，但是定位（找到第k个）慢。

课后习题与实验

学而时习之，不亦说乎。学而不思则罔，思而不学则殆。——孔子

请翻至课本 P220

复习

数组（vector）

给定下标，直接根据下标定位

栈（stack）

元素满足 后进先出 或者 先进后出，类似于洗盘子。
应用：后缀表达式、匹配括号等

队列（queue）

元素满足 先进先出 的性质，类似排队。
应用：广度优先搜索、秒杀商品等

链表（list）

知道每个元素前面一个和后面一个；中间插入删除效率高

作业

替换练习：

1. 尝试使用 C++ STL 中的栈，编写小止洗盘子的代码。
2. 对于给出的手写栈的代码，新增一个函数 int size()，查询栈内目前有多少元素。
3. 栈的手写方式多种多样。尝试：若更改栈顶指针 p 的定义，指向当前栈顶的位置，编写小止洗盘子的代码。

替换练习：

前缀表达式将运算符写在前面，操作数写在后面。例如 – 1 + 2 3 与 1-(2+3) 等价。请写出一个代码，可以输出给定的前缀表达式的值。

作业

习题 15.6 机器翻译（洛谷 P1540，NOIP2010 提高组）

软件内存容量为$M(M \leq 100)$。

每当软件将一个新单词存入内存前，如果内存未满，就会将新单词存入一个未使用的内存单元；否则软件会清空最早进入内存的那个单词，存放新单词。

一篇英语文章的长度为$N(N \leq 1000)$个单词（用数字编号代表每个单词），软件优先在内存中查找单词是否存在，如果不存在则需要联网查询并将新结果缓存进内存中。

给定这篇待译文章，翻译软件需要联网查询几次？假设在翻译开始前，内存中没有任何单词。

作业

习题 15.7 海港（洛谷 P2058，NOIP2016 普及组）

小 K 统计了 $n(n \le 10^5)$ 艘船的信息。

每行描述一艘船的信息：前两个整数 $t_i(t_i \le 10^9)$ 和 $k_i(\sum k_i < 3\times10^5)$ 分别表示这艘船到达海港的时间和船上的乘客数量，接下来 k_i 个整数 $x_{i,j}(x_{i,j} \le 10^5)$ 表示船上乘客的国籍。

你需要计算 n 条信息。对于输出的第 i 条信息，你需要统计满足 $t_i - 86400 < t_p \le t_i$ 的船只 p，在所有的 $x_{p,j}$ 中，总共有多少个不同的数。

作业

习题 15.9 验证栈序列（洛谷 P4387）

给出两个序列 pushed 和 poped 两个序列，其取值从 1 到 $n(n \leq 100000)$。已知入栈序列是 pushed，如果出栈序列有可能是 poped，则输出 Yes，否则输出 No。

为了防止骗分，每个测试点有多组数据。

参考阅读

以下的内容限于课件篇幅未能详细阐述。如果学有余力，可自行翻阅课本作为扩展学习。

例题 15.2：使用 vector 代替二维数组

例题 15.12：双向链表的基础应用

例题 15.13：使用链表解决约瑟夫问题，以及 STL 的迭代器

习题 15.1-15.5、15.8、15.10

课本上关于 vector 的迭代器以及 list 的应用。