









# STL 序列式容器研讨

经典算法设计技术研讨  
第二次讨论课（第4周）



# 目录

01/ 向量(vector)

02/ 列表(list)

03/ 双端队列(deque)

04/ 栈(stack)

05/ 队列(queue)





## PART.01

向量 (vector)



# 向量(vector)

## 一、概述

vector为可变长数组（动态数组），定义的vector数组可以随时添加数值和删除元素.

## 二、数据结构的定义

在局部函数中开vector数组，是在对空间里面开的，与开全局变量比较类似在向量中，所有数据项的物理存放位置与其逻辑次序完全吻合，此时的逻辑次序也称为秩(rank)。





## 向量 (vector)

### 三、关键基本操作

#### (1). 容量

向量大小: `vec.size();`

向量最大容量: `vec.max_size();`

更改向量大小: `vec.resize();`

向量真实大小: `vec.capacity();`

向量判空: `vec.empty();`

减少向量大小到满足元素所占存储空间的大小:

`vec.shrink_to_fit(); //shrink_to_fit`

#### (2). 修改

多个元素赋值: `vec.assign();` //类似于初始化时用数组进行赋值

末尾添加元素: `vec.push_back();`

末尾删除元素: `vec.pop_back();`

任意位置插入元素: `vec.insert();`

任意位置删除元素: `vec.erase();`

交换两个向量的元素: `vec.swap();`

清空向量元素: `vec.clear();`





## 向量 (vector)

### (3)迭代器

开始指针: `vec.begin();`

末尾指针: `vec.end();` //指向最后一个元素的下一个位置

指向常量的开始指针: `vec.cbegin();` //意思就是不能通过这个指针来修改所指的内容,但还是可以通过其他方式修改的,而且指针也是可以移动的。

指向常量的末尾指针: `vec.cend();`

### (4)元素的访问

下标访问: `vec[1];` //并不会检查是否越界

at方法访问: `vec.at(1);`

//以上两者的区别就是at会检查是否越界,是则抛出out of range异常

访问第一个元素: `vec.front();`

访问最后一个元素: `vec.back();`

返回一个指针: `int* p = vec.data();`

//可行的原因在于vector在内存中就是一个连续存储的数组,所以可以返回一个指针指向这个数组。这是C++11的特性。





## 向量 (vector)

举例说明1：遍历元素

```
vector<int>::iterator it;  
for (it = vec.begin(); it != vec.end(); it++)  
    cout << *it << endl;  
//或者  
for (size_t i = 0; i < vec.size(); i++) {  
    cout << vec.at(i) << endl;  
}
```

举例说明2：元素翻转

```
#include <algorithm>  
reverse(vec.begin(), vec.end());
```





## 向量 (vector)

例题：约瑟夫问题（大家都熟悉的很）

解题源码：

```
#include<iostream>
#include<vector>
using namespace std;
int main(){
    int n,k,m,x=0;
    vector<int> v;
    cin >> n >> k >> m;
    for(int i = 1; i <= n; i ++){
        v.push_back(i);
    }
    while(v.size()){
        x = (m+x-1)%v.size();
        cout << v[x] << endl;
        v.erase(v.begin()+x);
    }
}
```







**PQRT.02**

列表(list)



# 列表 (list)

## 一、概述

list适用于频繁的插入删除操作，对于随机存取需要遍历整个list

## 二、数据结构的定义

list在内存分配的时候不像vector一样分配连续的空间，只能使用迭代器遍历list的数据；（双向链表）

定义：list<数据类型> 变量名；//此处的变量类型可以为任意类型





## 列表 (list)

### 三、关键基本操作

正向迭代器: `begin()`和`end()`;

//`begin()`指向向量的第一个元素、`end()`指向向量的最后一个元素的下一个位置;

反向迭代器: `rbegin()`和`rend()`;

//`rbegin()`指向向量的最后一个元素、`rend()`指向第一个元素之前的位置;

`empty()`; //判断list是否为空;

`size()`; //返回list的实际大小;

`front()`; //获取list的第一个元素;

`back()`; //获取list最后一个元素;

`clear()`; //清除list中所有的元素;

`insert()`; //在list中插入数据; `insert(pos,num)` 在pos位置插入num;

`erase()`; //删除list中的元素; `erase(pos)` 删除list中pos位置的元素;





## 列表(list)

举例说明1：交换两个列表

```
list<int> a{6,7,8,9};  
list<int> b{1,2,3,4,5};  
swap(a, b); //或a.swap(b)
```

举例说明2：实现列表逆置

```
list<int> b{1,2,3,4,5};  
reverse(b.begin(),b.end());
```





## 列表 (list)

### 四、例题

第一行一个数字N，表示排队信息或者查询信息条目的数量。

以下N行，每行的内容有以下3种情况

- (1) in name 表示名字为name的人员新来到办事大厅，排在队伍的最后。  
(in和name间存在一个空格，name是名字对应字符串，长度不超过10)。
- (2) out 表示当前排在最前面的人已经办理完业务，离开了。
- (3) q 表示一次查询，请输出当前正在办理业务的人，也就是队伍的第1个人。如果当前无人办理业务，则输出“NULL”，不包括引号。

源码：

```
int main() {  
    list<string>l;  
    int n = 0;  
    cout << "输入多少人办理业务" << endl;  
    cin >> n;//表示多少n个人的业务  
    while (n) {  
        string s;  
        cout << "增加人按in,查询请按q,离开按out:" << endl;  
        cin >> s;
```






## 列表 (list)

```
if (s=="in") {
    cout << "请输入办理人姓名:" << endl;
    char a = getchar();//吸收空格
    getline(cin, s);
    l.emplace_back(s);
}

else if (s == "q" || s == "Q") {
    if (!l.empty())cout <<"排在第一位的人是:" << *l.begin() << endl;
    else {
        cout << "NULL" << endl;
    }
}

else if (s == "out") {
    if (!l.empty()) {
        l.erase(l.begin());
        --n;
    }
}
```





**PQRT.03**

双端队列 (deque)

# 双端队列 (deque)



## 一、概述

deque容器为一个给定类型的元素进行线性处理，像向量一样，它能够快速地随机访问任一个元素，并且能够高效地插入和删除容器的尾部元素。但它又与vector不同，deque支持高效插入和删除容器的头部元素，因此也叫做双端队列。

## 二、数据结构的定义

和 vector 容器采用连续的线性空间不同，deque 容器存储数据的空间是由一段一段等长的连续空间构成，各段空间之间并不一定是连续的，可以位于在内存的不同区域。







## 双端队列 (deque)

### 三、具体操作

#### 1插入操作:

`deque<string> dec;`

`void push_front(const T& x):` 双端队列头部增加一个元素X

`dec.push_front("world");` //头部插入

`void push_back(const T& x):`双端队列尾部增加一个元素x

`dec.push_back("hello");` //尾部插入

`iterator insert(iterator it,const T& x):`双端队列中某一元素前增加一个元素x

`dec.insert(dec.end(), "aaaaaa");`

`void insert(iterator it,int n,const T& x):`双端队列中某一元素前增加n个相同的元素x

`dec.insert(dec.end(), 5, "bbb");`





## 双端队列 (deque)

### 三、具体操作

#### 2、删除操作:

iterator erase(iterator it):删除双端队列中的某一个元素

```
dec.erase(dec.begin());
```

iterator erase(iterator first,iterator last):删除双端队列中[first,last) 中的元素

```
dec.erase(dec.end()-3, dec.end());
```

void pop\_front():删除双端队列中最前一个元素

```
dec.pop_front(); //头部删除
```

void pop\_back():删除双端队列中最后一个元素

```
dec.pop_back(); //尾部删除
```

void clear():清空双端队列

```
dec.clear(); //清空
```





## 双端队列 (deque)

### 四、例题

给定一个长度为  $n$  的数组  $num$  和滑动窗口的大小  $size$ ，找出所有滑动窗口里数值的最大值。

例如，如果输入数组 $\{2,3,4,2,6,2,5,1\}$ 及滑动窗口的大小 $3$ ，那么一共存在 $6$ 个滑动窗口，他们的最大值分别为 $\{4,4,6,6,6,5\}$ ；针对数组 $\{2,3,4,2,6,2,5,1\}$ 的滑动窗口有以下 $6$ 个： $\{[2,3,4],2,6,2,5,1\}$ ， $\{2,[3,4,2],6,2,5,1\}$ ， $\{2,3,[4,2,6],2,5,1\}$ ， $\{2,3,4,[2,6,2],5,1\}$ ， $\{2,3,4,2,[6,2,5],1\}$ ， $\{2,3,4,2,6,[2,5,1]\}$ 。

窗口大于数组长度或窗口长度为 $0$ 的时候，返回空。

数据范围： $0 \leq n \leq 10000$ ， $0 \leq size \leq 10000$ ，数组中每个元素的值满足  $0 \leq value \leq 10000$



# 双端队列 (deque)

```
#include<iostream>
using namespace std;
#include<queue>

vector<int> maxInWindows(const vector<int>& num, unsigned int size)
{
    vector<int> ret;
    int n = num.size();
    deque<int> dq;
    for (int i = 0; i < n; ++i) {
        while (!dq.empty() && num[dq.back()] < num[i]) {
            dq.pop_back();
        }
        dq.push_back(i);
        // 判断队列的头部的下标是否过期
        if (dq.front() + size <= i) {
            dq.pop_front();
        }
        // 判断是否形成了窗口
        if (i + 1 >= size) {
            ret.push_back(num[dq.front()]);
        }
    }
    return ret;
}

int main(){
    unsigned int size;int n;
    cin>>n>>size;
    vector<int>v;
    for(int i=0;i<n;i++){
        int m;
        cin>>m;
        v.push_back(m);
    }
    vector<int>res(n);
    res=maxInWindows(v,size);
    for(int i=0;i<res.size();i++){
        cout<<res[i]<<" ";
    }
    return 0;
}
```



**PQRT.04**

栈 (stack)

# 栈(stack)



## 一、概述

栈(stack)是一种后进先出的数据结构,即LIFO (last in first out),最后加入栈的元素将最先被取出来,在栈的顶端进行数据的插入与取出,这一端称为栈顶,相对的,把另一端称为栈底。

## 二、数据结构的定义

栈就是一个线性表,只不过,栈的Insert 和 delete只能在表尾。普通的线性表,在表中的任意位置都可以进行insert和delete操作。

LIFO: Last In First Out 后进先出,先进后出。栈顶(Top): 进行插入和删除操作的一端。栈底(Bottom)



# 栈 (stack)



## 三、关键操作

### 1、初始化一个栈: InitStack

//初始化一个栈

// @maxl: 代表 制定这个栈的最大可以存储多少个元素

//返回值: 返回初始化好的顺序栈指针

```
SeqStack * InitStack(int maxl)
```

```
{
```

```
    SeqStack *s = malloc(sizeof(*s));
```

```
    s->data = malloc(sizeof(SElemType)* maxl);
```

```
    s->top = -1;
```

```
    s->maxlen = maxl;
```

```
    return s;
```

```
}
```

### 2、销毁一个栈: DestroyStack

//销毁一个栈

```
void DestroyStack(SeqStack*s)
```

```
{
```

```
    if(s==NULL)
```

```
        return ;
```

```
    free(s->data);
```

```
    free(s);
```

```
}
```



## 栈 (stack)

### 三、关键操作

#### 3、清空一个栈: ClearStack

//清空一个栈

```
void ClearStack(SeqStack*s)
{
    if(s==NULL)
        return ;
    s->top=-1;
}
```

#### 4、判断一个栈是否为空

// 1 表示为空栈或不存在

// 0 表示非空栈

```
int StackIsEmpty(SeqStack*s)
{
    if(s==NULL || s->top
    ==-1 )
    {
        return 1;
    }
    return 0;
}
```



# 栈(stack)

## 四、例题

给定一个二维数组，0表示可以通行，1表示该处有墙不能通行，试问给定两点坐标，判断这两个点能不能相通。

```
int** map;//迷宫
int row;//迷宫行列(含边界)
int col;
pos now;//当坐标
int opt;//对位移偏量的选择
stack<pos> path;
int endrow;
int endcol;

};

void findPath()
{
    int xbegin,ybegin;
    cout<<"输入起点的行和列: "<<endl;
    cin>>xbegin>>ybegin;
    cout<<"输入终点的行和列: "<<endl;

    cin>>endrow>>endcol;
    endrow--;
    endcol--;
    //位移偏量
    pos setMove[4];
    setMove[0].posRow=0; setMove[0].posCol=1; //右移
    setMove[1].posRow=1; setMove[1].posCol=0; //下移
    setMove[2].posRow=0; setMove[2].posCol=-1; //左移
    setMove[3].posRow=-1; setMove[3].posCol=0; //上移

    //边界-1
    now.posRow=xbegin-1;
    now.posCol=ybegin-1;

    map[xbegin-1][ybegin-1]=1;

    while(now.posRow!=endrow||now.posCol!=endcol)
    {
        //先找每一步的怎么走
        int Row;
        int Col;
        while(opt<=3)
        {
            //移动
            Row=now.posRow+setMove[opt].posRow;
            Col=now.posCol+setMove[opt].posCol;

            //判断是否撞墙
            if(map[Row][Col]==0)
                break;
            opt++;//撞墙，换下一种方式
        }
        //如果能移动
        if(opt<=3)
        {
            cout<<"move:";
            cout<<Row+1<<","<<Col+1<<" ";
            //填入栈中
            path.push(now);
            //移动到下一个位置
            now.posRow=Row;
            now.posCol=Col;

            opt=0;
            //并堵住来的路
            map[Row][Col]=1;
        }
    }

    //如果不能移动
    else
    {
        //向后退
        if(path.empty())
        {
            cout<<"没有出路!"<<endl;
            return ;
        }
        //退回至上的一处
        pos next;//下一步的坐标
        next=path.top();
        cout<<"move:";
        cout<<next.posRow+1<<","<<next.posCol+1<<" ";
        path.pop();
        if(next.posRow==now.posRow)//行坐标相同
        {
            int lastmove=next.posCol-now.posCol;
            //1,上次左移
            //-1,上次右移
            opt=2+lastmove ;
        }
        else
        {
            int lastmove=next.posRow-now.posRow;
            opt=3+lastmove;
        }
        now=next;
    }
    cout<<"找到出路"<<endl;
}
```



**PQRT.05**

队列 (queue)



# 队列 (queue)

## 一、概述

队列是一种特殊的线性表，特殊之处在于它只允许在表的前端 (front) 进行删除操作，而在表的后端 (rear) 进行插入操作，和栈一样，队列是一种操作受限制的线性表。进行插入操作的端称为队尾，进行删除操作的端称为队头。队列中没有元素时，称为空队列。

## 二、数据结构的定义

建立顺序队列结构必须为其静态分配或动态申请一片连续的存储空间，并设置两个指针进行管理。一个是队头指针front，它指向队头元素；另一个是队尾指针rear，它指向下一个入队元素的存储位置。





## 队列(queue)

### 三、关键操作

#### 1、初始化队列:

//让队列的队头, 队尾分别指向空

```
void LQueueInit(LQueue *q)
{
    assert(q);
    q->pFront = q->pRear = NULL;
}
```

#### 2、元素入队:

```
void LQueuePush(LQueue *q, QElemType data)
{
    assert(q);
    if (NULL == q->pFront)
    {
        q->pFront = q->pRear = BuyLQNode(data);
        return;
    }
    q->pRear->_pNext = BuyLQNode(data);
    q->pRear = q->pRear->_pNext;
}
```





## 队列 (queue)

### 3、元素出队:

//这里的出队列采用是让队头元素不断后移，刷新队头元素，这样优化时间效率

```
void LQueuePop(LQueue *q)
{
    assert(q);
    QNode *pDel;
    if (NULL == q->pFront)
    {
        return;
    }
    if (q->pFront == q->pRear)
    {
        q->pRear = NULL;
    }
    pDel = q->pFront;
    q->pFront = q->pFront->_pNext;
    free(pDel);
}
```





## 队列 (queue)

### 四、例题

给定一个字符串  $s$ ，找到 它的第一个不重复的字符，并返回它的索引。如果不存在，则返回  $-1$ 。

示例 1:

输入:  $s = \text{"leetcode"}$

输出: 0

示例 2:

输入:  $s = \text{"loveleetcode"}$

输出: 2

示例 3:

输入:  $s = \text{"aabb"}$

输出: -1





```
class Solution {
```

```
public:
```

```
    int firstUniqChar(string s) {
```

```
        unordered_map<char, int> position; // 建立哈希映射，第一个数据类型为char，  
        第二个为int
```

```
        queue<pair<char, int>> q; // 把字符及其下标作为一个二元组放入队列
```

```
        int n = s.size();
```

```
        for (int i = 0; i < n; ++i) {
```

```
            if (!position.count(s[i])) { 函数count()，判断字符出现次数，字符已经出现过  
            返回1，没出现过返回0
```

```
                position[s[i]] = i; // 获得字符对应的下标
```

```
                q.emplace(s[i], i); // 函数emplace()，把字符和下标作为二元组存入队列
```

```
            }
```

```
            else {
```

```
                position[s[i]] = -1; // 字符出现两次
```

```
                while (!q.empty() && position[q.front().first] == -1) {
```

```
                    q.pop(); // 弹出队首元素
```

```
                }
```

```
            }
```

```
        }
```

```
        return q.empty() ? -1 : q.front().second; // 队列中字符全部重复，全部弹出，否则
```

```
        返回字符串中第一个唯一字符下标
```

```
    }
```

```
};
```





## 参考资料

- [1]<https://www.kancloud.cn/leehom/algorithmsummary/743544>[EB/OL].<https://www.jianshu.com/p/83dbb4199723>
- [2][https://blog.csdn.net/m0\\_37848958/article/details/80322709](https://blog.csdn.net/m0_37848958/article/details/80322709)  
[EB/OL].<https://zhidao.baidu.com/question/1518200437597767340.html>
- [3]<https://bbs.csdn.net/topics/396294632>
- [4]<https://wenku.baidu.com/view/bbb093756d85ec3a87c24028915f804d2b1687ab.html>
- [5]<https://www.csdn.net/tags/Ntjacg2sODIyNC1ibG9n.html>6]<https://bbs.csdn.net/topics/396294632>





THANKS

