数据结构

讨论课课前资料

题　　目：　第四次讨论课（第八周）

学生姓名：　魏子铖

学生学号：　201726010308

专业班级：　软件 1703

完成时间：　2018.11.6

一、 向量(vector)

1. 概述

vector 对应的数据结构为数组，而且是动态数组，也就是说我们不必关心该数组事先定义的容量是多少，它的大小会动态增长。与数组类似的是，我们可以在末尾进行元素的添加和删除，也可以进行元素值的随机访问和修改。首先要引入头文件 #include <vector>。

2. ADT

```
template <class T, classAlloc = alloc>
class vector {
    ...
    protected:
      iterator start; //表示目前使用空间的头
      iterator finish; //表示目前使用空间的尾
      iterator end_of_storage; //表示目前可用空间的尾
    ...
};
```

为了降低空间配置时的速度成 A，vector 实际配置的大小可能比客端需求量更大一些，以备将来可能的扩充。这便是容量（capacity）的观念。换句话说一个 vector 的容量永远大于或等于其大小。一旦容量等于大小，便是满载，下次再有新增元素，整个 vector 就得另觅居所。运用 start, finish, end_of_storage 三个迭代器，便可轻易提供首尾标示、大小、容量、空容器判断、注标（[ ]）运算子、最前端元素值、最后端元素值…等机能

```
template <class T, classAlloc = alloc>
class vector {
    ...
    public:
      iterator begin() { return start; }
      iterator end() { return finish; }
      size_typesize() const { return size_type(end() - begin()); }
      size_typecapacity() const {
      return size_type(end_of_storage - begin()); }
      bool empty() const { return begin() == end(); }
      referenceoperator[](size_type n) { return*(begin() + n); }
      referencefront() { return *begin(); }
      referenceback() { return *(end() - 1); }
    ...
};
```

3. 基本操作

c.assign(beg,end) 将(beg; end)区间中的数据赋值给 c。
c.assign(n,elem) 将 n 个 elem 的拷贝赋值给 c。
c. at(idx) 传回索引 idx 所指的数据。
c.back() 传回最后一个数据，不检查这个数据是否存在。
c.begin() 传回迭代器中的第一个数据地址。
c.capacity() 返回容器中数据个数。

c.clear()　移除容器中所有数据。
c.empty()　判断容器是否为空。
c.end()　指向迭代器中末端元素的下一个，指向一个不存在元素。
c.erase(pos)　删除 pos 位置的数据，传回下一个数据的位置。
c.erase(beg,end)　删除[beg,end)区间的数据。
c.front()　传回第一个数据。
get_allocator　使用构造函数返回一个拷贝。
c.insert(pos,elem)　在 pos 位置插入一个 elem 拷贝。
c.insert(pos,n,elem)　在 pos 位置插入 n 个 elem 数据,无返回值
c.insert(pos,beg,end)　在 pos 位置插入在[beg,end)区间的数据。
c.max_size()　返回容器中最大数据的数量。
c.pop_back()　删除最后一个数据。
c.push_back(elem)　在尾部加入一个数据。
c.rbegin()　传回一个逆向队列的第一个数据。
c.rend()　传回一个逆向队列的最后一个数据的下一个位置。
c.resize(num)　重新指定队列的长度。
c.reserve()　保留适当的容量。
c.size()　返回容器中实际数据的个数。
c1.swap(c2)　将 c1 和 c2 元素互换

4. 应用

【问题描述】

体育老师小明要将自己班上的学生按顺序排队。他首先让学生按学号从小到大的顺序排成一排，学号小的排在前面，然后进行多次调整。一次调整小明可能让一位同学出队，向前或者向后移动一段距离后再插入队列。小明记录了所有调整的过程，请问，最终从前向后所有学生的学号依次是多少?

请特别注意，上述移动过程中所涉及的号码指的是学号，而不是在队伍中的位置。在向后移动时，移动的距离不超过对应同学后面的人数，如果向后移动的距离正好等于对应同学后面的人数则该同学会移动到队列的最后面。在向前移动时，移动的距离不超过对应同学前面的人数，如果向前移动的距离正好等于对应同学前面的人数则该同学会移动到队列的最前面。

【输入格式】

输入的第一行包含一个整数 n，表示学生的数量，学生的学号由 1、到 n 编号。

第二行包含一个整数 m，表示调整的次数。

接下来 m 行，每行两个整数 p, q，如果 q 为正，表示学号为 p 的同学向后移动 q，如果 q 为负，表示学号为 p 的同学向前移动-q。

【输出格式】

输出一行，包含 n 个整数，相邻两个整数之间由一个空格分隔，表示最终从前向后所有学生的学号。

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

```
int main()
{
    int n,m,t=0,p,d;
    cin>>n>>m;
    int book[n];
    vector<int> vec;
    for(int i=1;i<=n;i++)
        vec.push_back(i);
    for(int i=0;i<n;i++)
        book[i]=i+1;
    while(t++<m)
    {
        cin>>p>>d;
        int point=0;
        for(int i=0;i<vec.size();i++)
            if(vec[i]==p)
                point=i;
        p--;
        int d_=book[p];
        int dis=point+d;
        if(d>0)
        {
            vec.insert(vec.begin()+dis+1,d_);
            vec.erase(vec.begin()+point);
        }
        if(d<0)
        {
            vec.erase(vec.begin()+point);
            vec.insert(vec.begin()+dis,d_);
        }
    }
    for(int i=0;i<vec.size();i++)
        cout<<vec[i]<<' ';
    return 0;
}
```

二、 列表(list)

1. 概述

list 是 C++标准模版库(STL,Standard Template Library)中的部分内容。实际上,list 容器就是一个双向链表,可以高效地进行插入删除元素。

使用 list 容器之前必须加上 STL 的 list 容器的头文件：#include<list>；

list 属于 std 命名域的内容，因此需要通过命名限定：using std::list；也可以直接使用全局的命名空间方式：using namespace std。

2. ADT

list 不仅是一个双向串行，而且还是一个环状双向串行。所以它只需要一个指标，便可以完整表现整个串行：

```
template <class T, classAlloc = alloc>// 预设使用 alloc 为配置器
class list {
protected:
 typedef__list_node<T>list_node;
public:
 typedef list_node* link_type;
protected:
 link_typenode;// 只要一个指标，便可表示整个环状双向串行
...
};
```

如果让指标 node 指向刻意置于尾端的一个空白节点，node 便能符合 STL 对于「前闭后开」区间的要求，成为 last 迭代器

```
iterator begin() { return (link_type)((*node).next); }
iterator end() { return node; }
bool empty() const { return node->next == node; }
size_typesize() const {
size_type result = 0;
distance(begin(), end(), result);
    return result;
 }
 // 取头节点的内容（元素值）。
 referencefront() { return *begin(); }
 // 取尾节点的内容（元素值）。
referenceback() { return *(--end()); }
```

3. 基本操作

```
assign() //分配值，有两个重载：
c1.assign(++c2.begin(), c2.end()) //c1 现在为(50,60)。
c1.assign(7,4) //c1 中现在为 7 个 4,c1(4,4,4,4,4,4,4)。
back() //返回最后一元素的引用：
begin() //返回第一个元素的指针(iterator)
clear() //删除所有元素
empty() //判断是否链表为空
end() //返回最后一个元素的下一位置的指针(list 为空时 end()=begin())
erase() //删除一个元素或一个区域的元素(两个重载)
front() //返回第一个元素的引用：
insert() //在指定位置插入一个或多个元素(三个重载)：
max_size() //返回链表最大可能长度(size_type 就是 int 型)：
merge() //合并两个链表并使之默认升序(也可改)：
pop_back() //删除链表尾的一个元素
pop_front() //删除链表头的一元素
push_back() //增加一元素到链表尾
push_front() //增加一元素到链表头
```

rbegin() //返回链表最后一元素的后向指针(reverse_iterator or const)
rend() //返回链表第一元素的下一位置的后向指针
remove //()删除链表中匹配值的元素(匹配元素全部删除)
remove_if() //删除条件满足的元素(会遍历一遍链表)
resize() //重新定义链表长度(两重载):
reverse() //反转链表:
size() //返回链表中元素个数
sort() //对链表排序，默认升序(可自定义)
splice() //对两个链表进行结合(三个重载)
swap() //交换两个链表(两个重载)
unique() //删除相邻重复元素(断言已经排序，因为它不会删除不相邻的相同元素)

4. 应用

【问题描述】

一个多项式可以表达为 x 的各次幂与系数乘积的和，比如:

现在，你的程序要读入两个多项式，然后输出这两个多项式的和，也就是把对应的幂上的系数相加然后输出。

【输入格式】

总共要输入两个多项式，每个多项式的输入格式如下:

- 每行输入两个数字，第一个表示幂次，第二个表示该幂次的系数
- 以 0 0 结束一个多项式的输入
- 注意第一行和最后一行之间不一定按照幂次降低顺序排列
- 如果某个幂次的系数为 0，就不出现在输入数据中了
- 0 次幂的系数为 0 时还是会出现在输入数据中

【输出格式】

从最高幂开始依次降到 0 幂，如:

2x^6+3x^5+12x^3-6x+20

注意其中的 x 是小写字母 x，而且所有的符号之间都没有空格，如果某个幂的系数为 0 则不需要有那项。

```
#ifndef POLYNOMIAL_POLY_H
#define POLYNOMIAL_POLY_H

#include <iostream>
struct Node {
    int index;
    double coef;
    Node* next;

    explicit Node(double c, int i, Node* ptr = NULL);
    explicit Node(Node* ptr = NULL);
};

class Poly {
```

```
private:
    Node* head;
    Node* curr;
    Node* tail;

    void init() {
        curr = tail = head = new Node;
        tail = new Node(-1, -1);
        head -> next = tail;
    }
    void removeAll() {
        while (head != NULL) {
            curr = head;
            head = head -> next;
            delete curr;
        }
    }

public:
    Poly();
    ~Poly();
    void insert(const double&, const int&);
    void next();
    bool hasNext();
    void clear();
    void print() const;

    double getCoef();
    int getIndex();

};

#endif //POLYNOMIAL_POLY_H

#include "Poly.h"

Node::Node(double c, int i, Node *ptr) {
    coef = c;
    index = i;
    next = ptr;
}

Node::Node(Node *ptr) {
    index = 2147483647;
```

```
	coef = 999999;
	next = ptr;
}

Poly::Poly() {
	init();
}

Poly::~Poly() {
	removeAll();
}

void Poly::insert(const double & coef, const int & index) {
	Node* tmp = head;
	if (tmp -> next == tail)
		head -> next = new Node(coef, index, tail);
	else {
		while (tmp -> next -> index > index) {
			tmp = tmp -> next;
		}
		tmp -> next = new Node(coef, index, tmp -> next);
	}
}

void Poly::next() {
	if (curr -> next != tail)
		curr =	curr -> next;
}

bool Poly::hasNext() {
	return curr -> next != tail;
}

void Poly::clear() {
	removeAll();
	init();
}

double Poly::getCoef() {
	return curr -> next -> coef;
}

int Poly::getIndex() {
	return curr -> next -> index;
```

```
}

void Poly::print() const {
    Node* tmp = head;
    std::cout << "A(x) + B(x) = ";
    if ((int)tmp -> next -> coef == 1) {
        if (tmp -> next -> index == 0)
            std::cout << '1';
    }
    else if ((int)tmp -> next -> coef == -1) {
        std::cout << '-';
    }
    else {
        std::cout << tmp -> next -> coef;
    }
    if (tmp -> next -> index != 0) {
        if (tmp -> next -> index == 1) {
            std::cout << "x";
        }
        else {
            std::cout << "x^"
                      << tmp -> next -> index;
        }
    }
    tmp = tmp -> next;
    while (tmp -> next != tail) {
        if (tmp -> next -> coef == 1) {
            if (tmp -> next -> index == 0)
                std::cout << "+1";
            else
                std::cout << '+';
        }
        else if ((int)tmp -> next -> coef == -1) {
            std::cout << '-';
        }
        else {
            if (tmp -> next -> coef > 0) {
                std::cout << '+';
                if (tmp -> next -> index == 0 && tmp -> next -> coef == 1)
                    std::cout << '1';

            }
            std::cout << tmp -> next -> coef;
        }
```

```
            if (tmp -> next -> index != 0) {
                if (tmp -> next -> index == 1) {
                    std::cout << "x";
                }
                else {
                    std::cout << "x^"
                              << tmp -> next -> index;
                }
            }
            tmp = tmp -> next;
        }
        std::cout << std::endl;
}

#include "Poly.h"

int main() {
        double coef = 0.0;
        int index = 0;
        Poly A, B, C;
        std::cout << "每行输入两个数字，第一个表示幂次，第二个表示该幂次的
系数，以空格分离\n"
                     "以 0 0 结束一个多项式的输入\n"
                  << "===输入多项式 A(x)==="
                  << std::endl;
        while (std::cin >> index >> coef) {
            if ((int)coef == 0 && index == 0) {
                break;
            }
            A.insert(coef, index);
        }
        std::cout << "===输入多项式 B(x)==="
                  << std::endl;
        while (std::cin >> index >> coef) {
            if ((int)coef == 0 && index == 0) {
                break;
            }
            B.insert(coef, index);
        }

        while (A.hasNext() || B.hasNext()) {
            if (A.hasNext() && B.hasNext()) {
                int indexA = A.getIndex();
                int indexB = B.getIndex();
```

```
            if (indexA == indexB) {
                double coefA = A.getCoef();
                double coefB = B.getCoef();
                coef = coefA + coefB;
                if ((int)coef != 0)
                    C.insert(coef, indexA);
                A.next();
                B.next();
            }
            else if (indexA > indexB) {
                coef = A.getCoef();
                C.insert(coef, indexA);
                A.next();
            }
            else {
                coef = B.getCoef();
                C.insert(coef, indexB);
                B.next();
            }
        }
        else if (A.hasNext()) {
            int indexA = A.getIndex();
            coef = A.getCoef();
            C.insert(coef, indexA);
            A.next();
        }
        else {
            int indexB = B.getIndex();
            coef = B.getCoef();
            C.insert(coef, indexB);
            B.next();
        }
    }
    A.clear();
    B.clear();
    C.print();
    system("pause");
    return 0;
}
```

三、 双端队列(deque)

1. 概述

#include <deque>deque 容器类与 vector 类似，支持随机访问和快速插入删除，它在容器中某一位置上的操作所花费的是线性时间。与 vector 不同的是，deque 还支持从开始端插入数据：push_front()。

2. **ADT**

```
template <class T, classAlloc = alloc, size_t BufSiz = 0>
class deque {
public: // Basic types
typedef Tvalue_type;
 typedef value_type*pointer;
 typedef size_t size_type;
public: // Iterators
 typedef__deque_iterator<T, T&, T*,BufSiz> iterator;
protected: // Internal typedefs
// 元素的指针的指针（pointer of pointer of T）
 typedef pointer* map_pointer;
protected: // Data members
iterator start;
iterator finish;
 map_pointermap;
//表现第一个节点。
//表现最后一个节点。
//指向 map，map 是块连续空间，
 // 其每个元素都是个指针，指向一个节点（缓冲区）。
 size_typemap_size;// map 内有多少指标。
...
};
public: // Basic accessors
iterator begin() { return start; }
iterator end() { return finish; }
          reference operator[](size_type n) {
               return start[difference_type(n)];
//唤起__deque_iterator<>::operator[]
          }
reference front(){ return *start; }
// 唤起 __deque_iterator<>::operator*
 reference back() {
 iterator tmp = finish;
 --tmp;//唤起 __deque_iterator<>::operator--
 return *tmp; //唤起 __deque_iterator<>::operator*
}
size_typesize() const { return finish - start;; }
// 以上唤起 iterator::operator-
size_typemax_size() const { return size_type(-1); }
bool empty() const { return finish == start; }
```

3. **基本操作**

c.assign(beg,end) 将[beg; end)区间中的数据赋值给 c。
c.assign(n,elem) 将 n 个 elem 的拷贝赋值给 c。

c. at(idx) 传回索引 idx 所指的数据，如果 idx 越界，抛出 out_of_range。
c.back() 返回容器 c 最后一个元素的引用。如果 c 为空，则该操作未定义。
c.begin() 传回迭代器中的第一个数据地址。
c.clear() 移除容器中所有数据。
c.empty() 判断容器是否为空。
c.end() 返回一个迭代器，它指向容器 c 的最后一个元素的下一位置。
c.erase(pos) 删除 pos 位置的数据，传回下一个数据的位置。
c.erase(beg,end) 删除[beg,end)区间的数据，传回下一个数据的位置。
c.front() 返回容器 c 的第一个元素的引用。如果 c 为空，则该操作为空。
get_allocator 使用构造函数返回一个拷贝。
c.insert(pos,elem) 在 pos 位置插入一个 elem 拷贝，传回新数据位置
c.insert(pos,n,elem) 在 pos 位置插入>n 个 elem 数据。无返回值
c.insert(pos,beg,end) 在 pos 位置插入在[beg,end)区间的数据。无返回值
c.max_size() 返回容器 c 可容纳的最多元素个数。
c.pop_back() 删除最后一个数据。
c.pop_front() 删除头部数据。
c.push_back(elem) 在尾部加入一个数据。
c.push_front(elem) 在头部插入一个数据。
c.rbegin() 返回一个逆序迭代器，它指向容器 c 的最后一个元素。
c.rend() 返回一个逆序迭代器，它指向容器 c 的第一个元素的前一个位置。
c.resize(num) 重新指定队列的长度。
c.size() 返回容器中实际数据的个数。
c.swap(c2) 交换容器 c 和 c2 中的所有元素。
swap(c1,c2) 交换容器 c1 和 c2 中的所有元素，和上一方法相似。

4. 应用

HDUOJ6375 双端队列

```
#include <bits/stdc++.h>
using namespace std;
const int N = 150005;
map<int, deque<int> > q;
      void read(int &x){
      char ch = getchar();x = 0;
      for (; ch < '0' || ch > '9'; ch = getchar());
      for (; ch >='0' && ch <= '9'; ch = getchar()) x = x * 10 + ch
- '0';
}
int main()
{
    int a,n,m,u,v,w,val;
       while(~scanf("%d%d",&n,&m))
       {
            for(int i = 1; i <= n; i++)
                 q[i].clear();
            while(m--)
```

```
{
    read(a);
    if(a == 1)
    {
        read(u);
        read(w);
        read(val);
        if(w == 0) q[u].push_front(val);
        else if(w == 1) q[u].push_back(val);
    }
    else if(a == 2)
    {
        read(u);
        read(w);
        if(q[u].empty())
        {
            puts("-1");
            continue;
        }
        if(w == 0)
        {
            printf("%d\n",q[u].front());
            q[u].pop_front();
        }
        else if(w == 1)
        {
            printf("%d\n",q[u].back());
            q[u].pop_back();
        }
    }
    else if(a == 3)
    {
        read(u);
        read(v);
        read(w);
        if(w == 0)
        {
            q[u].insert(q[u].end(),q[v].begin(),q[v].end());
            q[v].clear();
        }
        else if(w == 1)
        {
            q[u].insert(q[u].end(),q[v].rbegin(),q[v].rend());
            q[v].clear();
```

```
                    }
                }
            }
        }
        return 0;
}
```

四、 栈(stack)

1. 概述

栈（stack）在计算机科学中是限定仅在表尾进行插入或删除操作的线性表。栈是一种数据结构，它按照后进先出的原则存储数据，先进入的数据被压入栈底，最后的数据在栈顶，需要读数据的时候从栈顶开始弹出数据。栈是只能在某一端插入和删除的特殊线性表。用桶堆积物品，先堆进来的压在底下，随后一件一件往上堆。取走时，只能从上面一件一件取。读和取都在顶部进行，底部一般是不动的。栈就是一种类似桶堆积物品的数据结构，进行删除和插入的一端称栈顶，另一端称栈底。插入一般称为进栈，删除则称为退栈。 栈也称为后进先出表。

2. ADT

```
typedef struct {
    SLink top; //栈顶指针
     int length; //  栈中元素个数
}Stack;
void InitStack (Stack &S);
bool Push (Stack &S, ElemType e);
bool Pop (Stack &S, SElemType &e);
```

3. 基本操作

empty() 堆栈为空则返回真
pop() 移除栈顶元素（不会返回栈顶元素的值）
push() 在栈顶增加元素
size() 返回栈中元素数目
top() 返回栈顶元素

4. 应用

利用 STL 栈将中缀表达式转换成后缀表达式输出

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;
int isp(char ch)
{
    switch(ch)
    {
        case'#':return 0;
        case'(':return 1;
        case'*':
```

```
                case'/':
                case'%':
                        return 5;
                case'+':
                case'-':
                                return 3;
                case')':
                        return 6;
                }
        }
        int icp(char ch)
        {
                switch(ch)
                {
                        case'#':return       0;
                        case'(':return 6;
                        case'*':
                        case'/':
                        case'%':
                                return 4;
                        case'+':
                        case'-':
                                        return 2;
                        case')':
                                return 1;
                }
        }
        int main()
        {
                stack<string> x;
                stack<char>     s;
                char ch1='#';
                double operand;
                s.push(ch1);
                cin.get(ch1);
                while(!s.empty()&&ch1!='#')
                {
                        if(isdigit(ch1))
{cin.putback(ch1);cin>>operand;(cout<<operand<<"                ";cin.get(ch1);}
//判断是否是数字，若是数字，则输出
                        else {
                                if(icp(ch1)>isp(s.top())) {s.push(ch1);cin.get(ch1);}
        //如果栈外 icp>栈内 isp  ，压栈且读入下一个字符
                                else    if(icp(ch1)<isp(s.top())){cout<<s.top()<<"    ";s.pop();}
```

```
//如果栈外 icp<栈内 isp，输出栈顶元素并退栈
                    else if(icp(ch1)==isp(s.top())){
    //如果 icp>isp，1--如果栈顶元素为"(",则退栈，并读入下一个元素

// 2--如果栈顶元素不为"("，仅仅退栈;
                        if(s.top()=='(') cin.get(ch1);
                        s.pop();
                    }
                }
            }
            cout<<"循环结束"<<endl;
                //cout<<s.top()<<endl;
                cin.get();
            return 0;
}
```

五、 队列(queue)

1. 概述

queue 是一种先进先出（First In First Out，FIFO）的数据结构。它有两个出口。queue 允许新增元素、移除元素、从最底端加入元素、取得最顶端元素。但除了最底端可以加入、最顶端可以取出，没有任何其它方法可以存取 queue 的其它元素。换言之 queue 不允许有走访行为。

2. ADT

```
typedef struct {
    int *base; // 初始化的动态分配存储空间
    int front; // 头指针，若队列不空，指向队列头元素
    int rear; // 尾指针，指向队列尾元素的下一个位置
} SqQueue;
*SqQueue Q_Init();
void Q_Destroy(SqQueue *Q);
void Q_Clear(SqQueue *Q);
int Q_Empty(SqQueue Q)
int Q_Length(SqQueue Q)
int Q_GetHead(SqQueue Q, int &e)
void Q_Print(SqQueue Q)
int Q_Put(SqQueue *Q, int e)
int Q_Poll(Queue *Q)
```

3. 基本操作

back()返回最后一个元素
empty()如果队列空则返回真
front()返回第一个元素
pop()删除第一个元素
push()在末尾加入一个元素
size()返回队列中元素的个数

4. 应用

```
#include <stdio.h>
#include <queue>
#include <algorithm>
using namespace std;

int main(int argc, char** argv)
{
    //deque 没有迭代器，stack 也没有，因此都不能排序
    //deque 插入元素只能从尾部插入，弹出智能从头部弹出
    //创建单向队列
    queue<int> q;
    printf("size(): %d\n", q.size());
    printf("empty(): %d\n", q.empty());
    //队尾插入元素
    q.push(1);
    q.push(2);
    q.emplace(3);
    //从单向队列头部弹出元素
    q.pop();
    //返回队首元素
    printf("front(): %d\n", q.front());
    //返回队尾元素
    printf("back(): %d\n", q.back());
    printf("size(): %d\n", q.size());
    printf("empty(): %d\n", q.empty());

    //交换元素
    queue<int> qTemp;
    qTemp.swap(q);
    //返回队首元素
    //printf("front(): %d\n", q.front());     //出错
    //返回队尾元素
    //printf("back(): %d\n", q.back());           //出错
    //返回队首元素
    printf("front(): %d\n", qTemp.front());
    //返回队尾元素
    printf("back(): %d\n", qTemp.back());
    getchar();
    return 0;
}
```

参考文献

1. 百度百科：词条 STL vector
   https://baike.baidu.com/item/STL%20vector/863812?fr=aladdin#1
2. 百度百科：词条 STL list
   https://baike.baidu.com/item/STL%20List/862006?fr=aladdin
3. C++的 STL 中向量（vector）的使用说明 –CSDN 博客
   https://blog.csdn.net/u011499425/article/details/52603921
4. 百度百科：词条 STL deque
   https://baike.baidu.com/item/STL%20deque
5. 利用 STL 栈将中缀表达式转换成后缀表达式输出 –CSDN 博客
   https://blog.csdn.net/qq1169091731/article/details/51112201