《数据结构与算法》实验报告

实验题目	线性数据结构实践		
实验时间	2023 年 10 月 19 日 19:00-22:00	实验地点	DS3402
实验成绩		实验性质	☐验证性 √设计性 ☐综合性

教师评价：

☐算法/实验过程正确； ☐源程序/实验内容提交 ☐程序结构/实验步骤合理；

☐实验结果正确； ☐语法、语义正确； ☐报告规范；

其他：

评价教师签名：

一、实验目的

1. 掌握线性表的基本原理及基本操作方法
2. 训练使用线性表并设计算法，编程解决不同难度问题的实践能力

二、实验项目内容

注：每道题按下面的格式分别描述

实验题目 1：模拟队列

题目内容：

实现一个队列，队列初始为空，支持四种操作：

1.push x – 向队尾插入一个数 x；
2.pop – 从队头弹出一个数；
3.empty – 判断队列是否为空；
4.query – 查询队头元素。

现在要对队列进行 M 个操作，其中的每个操作 3 和操作 4 都要输出相应的结果。

输入格式:

第一行包含整数 M，表示操作次数。

接下来 M 行，每行包含一个操作命令，操作命令为 push x，pop，empty，query 中的一种。

报告创建时间：

输出格式:

对于每个 empty 和 query 操作都要输出一个查询结果，每个结果占一行。

其中，empty 操作的查询结果为 YES 或 NO，query 操作的查询结果为一个整数，表示队头元素的值。

数据范围：

1≤M≤100000,
1≤x≤109,
所有操作保证合法。

代码：

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

int main() {
    int N;
    cin >> N;
    vector<int> que;
    int head = 0;
    int tail = 0;
    for (int i = 0; i < N; i++) {
        string s;
        cin >> s;
        if (s == "push") {
            int k;
            cin >> k;
            que.push_back(k);
            tail++;
        }
        else if (s == "empty") {
            if (head==tail)
```

```
                cout << "YES" << endl;
            else
                cout << "NO" << endl;
        }
        else if (s == "query") {
            cout << que[head]<<endl;
        }
        else if (s == "pop") {
            que[head] = 0;
            head++;
        }
        else {}
    }
}
```

实验题目 2：括弧匹配检验(check)

题目内容：

假设表达式中允许包含两种括号：圆括号和方括号，其嵌套的顺序随意，如 ([ ] ( )) 或 [([ ] [ ])] 等为正确的匹配，[( ]) 或 ([ ] ( ) 或 (( ))) 均为错误的匹配。

现在的问题是，要求检验一个给定表达式中的括弧是否正确匹配？

输入一个只包含圆括号和方括号的字符串，判断字符串中的括号是否匹配，匹配就输出 OK，不匹配就输出 Wrong。

输入格式:

输入仅一行字符（字符个数小于 255）。

输出格式:

匹配就输出 OK，不匹配就输出 Wrong。

代码：

```
#include <iostream>
#include <string>
```

```
#include <stack>
using namespace std;

bool isleft(char c) {
	if (c == '(' || c == '[')
		return true;
	else return false;
}

bool isright(char c) {
	if ( c == ')'    || c == ']')
		return true;
	else return false;
}

bool iscorrect(char c1, char c2) {
	if (c1 == '[' && c2 == ']')  return true;
	else if (c1 == '(' && c2 == ')') return true;
	else return false;
}



int main() {
	bool flag;
	string s;cin >> s;
	stack<char>bracket;
	int i = 0;
	flag = 1;
	while (s[i] != '\0' && flag) {
		if (isleft(s[i])) {
			bracket.push(s[i]);
```

```
            }
            else if (isright(s[i]) && !bracket.empty()) {
                if (iscorrect(bracket.top(), s[i])) bracket.pop();
            }
            else flag = 0;
            i++;
        }
        if (flag && bracket.empty())cout << "OK\n";
        else cout << "Wrong\n";
    }
```

实验题目 3：胡同

题目内容：

有一个死胡同，宽度刚好只能让一辆汽车通过，偏偏老有汽车开到死胡同来，这下麻烦了，最先开来的汽车要最后才能倒退出去。给定一个汽车开来的序列和一个可能的倒车出去的序列，请判断汽车能否都倒退出去，若能则输出 Yes，否则输出 No。

输入格式:

首先输入一个整数 T，表示测试数据的组数，然后是 T 组测试数据。每组测试数据首先输入一个正整数 n(n≤10)，代表开来的汽车数，然后输入 2n 个整数，其中，前 n 个整数表示汽车开来的序列，后 n 个整数表示汽车可能倒出的序列。

输出格式:

对于每组测试，判断能否倒车出该死胡同，若能则输出 Yes，否则输出 No。

代码：

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

int main() {
```

```
        int Nline; cin >> Nline;
        for (int i0 = 0; i0 < Nline;i0++){
            int N; cin >> N;
            int* carin = new int [N];
            int* carout = new int [N];
            stack<int>sta;
            for(int i = 0;i < N;i++) { cin >> carin[i]; }
            for(int i = 0;i < N;i++) { cin >> carout[i]; }
            //for (int i = 0; i < N; i++) { cout << "a" << carin[i] << " b" << carout[i] <<
endl; }
            int a = 0, b = 0;//a 进去 b 出来
            while(b < N) {
                if(carin[a] == carout[b]) {
                    a++;
                    b++;
                    //cout << "direct out" << carin[a - 1]<<endl;
                }
                else if(!sta.empty() && sta.top() == carout[b]) {
                    b++;
                    sta.pop();
                    //cout << "out stack" << carout[b - 1]<<endl;
                }
                else if(a < N) {
                    sta.push(carin[a]);
                    a++;
                    //cout << "in stack" << carin[a - 1]<<endl;
                }
                else break;
            }
            if(b == N&&sta.empty())cout << "Yes"<<endl;
            else cout << "No"<<endl;
```

```
    }
}
```

实验题目 4：列车车厢重排

题目内容：

火车站的列车调度铁轨的结构如下所示： (Exit) 9 8 7 6 5 4 3 2 1 <==== …… <==== 8 4 2 5 3 9 1 6 7 (Entrance) 两端分别是一条入口（Entrance）轨道和一条出口（Exit）轨道，它们之间可能有 N 条平行的轨道。每趟列车从入口可以选择任意一条轨道进入排队，以方便最后有序从出口离开。在前例中有 9 趟列车，在入口处按照{8，4，2，5，3，9，1，6，7}的顺序排队等待进入。如果要求它们必须按序号递减的顺序从出口离开，则至少需要多少条平行铁轨用于调度？调度入队后，各个队列里车厢情况如何？

输入格式：

输入第一行给出一个整数 N (2 ≤ N ≤99)，下一行给出从 1 到 N 的整数序号的一个重排列。数字间以空格分隔。

输出格式：

第一行输出 1 号车厢所在的队列中的元素（车厢编号间以空格分隔），注意，调度时，车厢只进入队列等待，并不出队。

在第二行中输出可以将输入的列车按序号递减的顺序重排所需要最少的辅助铁轨（队列）条数。

代码：

```
#include<iostream>
#include<queue>
using namespace std;
int main() {
    int N;
    cin >> N;
    int* tin = new int[N];//in:数组；out:N 到 1
    int tout =N;
    for(int i = 0;i < N;i++) {
        cin >> tin[i];
    }
    queue<int>* q = new queue<int>[N];
```

```
        int lines = 1;
        int linefor1 = 0;
        q[0].push(tin[0]);
        if(tin[0] == 1)linefor1 = 0;
        for(int i = 1;i < N;i++) {
            int minline = 0;
            int minofbigger = N;
            bool canpush = 0;
            for(int j = 0;j < lines;j++) {
                /*cout << "q" << j << ": " << q[j].back() << "\t";
                cout << "tin[i]: " << tin[i] << endl;*/
                if(q[j].back() <= minofbigger && q[j].back() > tin[i]) { canpush =
1;minline = j;minofbigger = q[j].back(); }
            }
            if(!canpush) {
                q[lines].push(tin[i]);
                lines++;
                if(tin[i] == 1)linefor1 = lines;
            }
            else{
            q[minline].push(tin[i]);
            if(tin[i] == 1)linefor1 = minline;
            }
        }
        cout << q[linefor1].front();
        q[linefor1].pop();
        while(!q[linefor1].empty()) {
            cout << " " << q[linefor1].front();
            q[linefor1].pop();
        }
        cout << "\n" << lines;
```

}

实验题目 5：倍数对

题目内容：

对于一个整型数组(p1 ,p2 ,…,pn)，如果某个下标小的元素是下标大的元素的倍数，我们称这两个元素为倍数对。形式化地，若存在 i 和 j，满足 i<j 且 pi 是 pj 的倍数，则称(pi ,pj)为倍数对。给定一个整型数组，数组元素是整数 1…n 的一个排列组合，请编写程序，计算该数组中所有倍数对的数目。

输入格式:

每个测试点包含多组数据。第一行是整数 T(1≤T≤104)，表示测试数据组数。对于每组数据，第一行为整数 n(1≤n≤105)，表示给定数组的长度，第二行为 n 个空格间隔的整数，表示给定的数组，保证给定的数组是整数 1…n 的一个排列组合。保证∑n≤106。

输出格式:

对每组数据输出一行，为一个整数，表示数组中倍数对的数目。

代码：

```
#include<iostream>
using namespace std;
int main() {
    int T; cin >> T;
    for (int i0 = 0; i0 < T; i0++) {
        int n; cin >> n;
        int count=0;
        int* a = new int[n];
        int* c = new int[n];
        for (int i = 0; i < n; i++) {
            int b;
            cin >> b;
            a[i] = b;
            c[b] = i;
        }
        int num = 1;
```

```
            while(num<n/2+1){
                int i = 2;
                while (i * num <= n) {
                    if (c[num] > c[i * num])count++;
                    i++;
                }
                num++;
            }
            cout << count << endl;
        }
    }
```