上 海 大 学

2021-2022 冬季学期

《数据结构（1）》实验报告

实 验 组 号：08

上 课 老 师：沈 俊

小 组 成 绩：

小组成员成绩表

序号	学号	姓名	贡献因子	成绩
1	20120500	王静颐	20	
2	20120796	康高熙	20	
3	20121034	胡才郁	20	
4	20121076	刘元	20	
5	20124633	金之谦	20	

注：小组所有成员的贡献因子之和为 100.

计算机工程与科学学院

2021 年 12 月 28 日

实验三 栈和队列

1 设计性实验

1.1 设计2个顺序栈共享存储空间的类模板

1.1.1 基本功能介绍

头对头双栈系统本质上是一个数组，但是可以选择从其头部或尾部按照栈的方式添加元素，也就是说，两个栈的长度相加最大不可以超过数组的长度。但是在不超过的情况下，两个栈的长度是可以一定程度上动态变化的。

1.1.2 主要算法设计

选择栈函数——getID()

为了防止键盘输入的非法字符使程序崩溃，特意设计了函数，从键盘读入数字而不至于是程序崩溃

```
template<class T>
int DoublyStack<T>::getID(){
    int id;
    std::cout << "input a id(0 for the 1st, 1 for the 2nd): ";
    while (!(std::cin >> id) || !(judge_id(id))){
        std::cout << "illegal ID, input again: ";
        std::cin.clear();
        std::cin.ignore(1024, '\n');
    }
    return id;
}
```

图 1. getID()

构造函数

构造函数并没有太多值得赘述的东西，但是系统本身的设计借鉴了 STL 的思想，非常巧妙。定义了两个 last 指针指向了数组两边的内存，但是并不对其进行取值操作，仅仅记录其地址，方便比较操作。

```
template<class T>
DoublyStack<T>::DoublyStack(int size){
    if (size <= 0)
        throw size;
    memset(firstPtr, 0, sizeof(firstPtr));
    memset(lastPtr, 0, sizeof(lastPtr));
    this->_size = size;
    this->_storage = new T[size];
    this->_lLast = _storage - 1;
    this->_rLast = _storage + this->_size;
    this->_lFirst = this->_lLast;
    this->_rFirst = this->_rLast;
}
```

图 2.构造函数

基本添加功能——push(int id, T data)

基于上述的设计，在添加的时候，就可以做到先挪动 first 指针，然后判断储存空间是否已满，再进行下一步操作

```
/************************full()*********************/
inline bool full() { return _lFirst + 1 == _rFirst; }

/************push(int id, T data)*****************/
template<class T>
void DoublyStack<T>::push(int id, T data){
    if (full()){
        std::cout << "the internal storage is full, push fail" << std::endl;
        return;
    }
    switch (id){
    case 0:
        _lFirst++;
        *_lFirst = data;
        break;
    case 1:
        _rFirst -= 1;
        *_rFirst = data;
    default:
        break;
    }
}
```

图 3. InsertElem 功能函数

1.1.3 主要数据组织

在原有模板的基础上，添加了拷贝构造函数与赋值运算符函数的设计，并且实现了基本的增删改查功能。这里主要对其整体结构进行一个梳理：

表 1. CircularLinkList 类中各函数的声明

函数原型	返回值类型	功能
CircularLinkList();		空参构造函数，初始化链表（空链表）
CircularLinkList(T v[], int size);		初始化链表
virtual ~CircularLinkList();		析构函数，清空链表
CircularLinkList(const CircularLinkList<T> &l);		拷贝构造函数
CircularLinkList<T> &operator= (const CircularLinkList<T> &l);	CircularLinkList<T>&	赋值运算符函数
void Clear(); // 清空循环单链表	void	清空链表
void Show() const;	void	输出链表信息
int LocateElem(const T &e) const;	int	按值查找
Status GetElem(int i, T &e) const;	Status	按位查找
Status SetElem(int i, const T &e);	Status	修改指定位置元素值

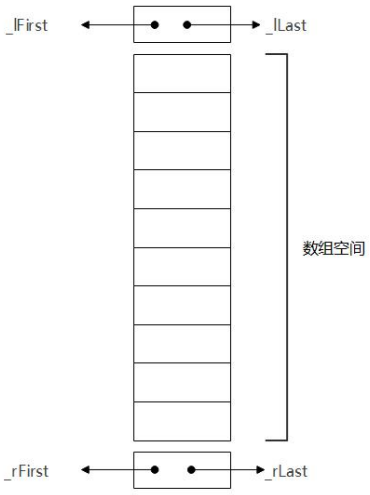
Status DeleteElem(int i, T &e);	Status	删除指定元素
Status InsertElem(int i, const T &e);	Status	插入指定元素

以下介绍部分函数以及设计思路：

1.1.4 测试分析

双向栈测试菜单如下，实现了与用户的交互，并且处理了输入英文字符等非法输入的非法情况。

```
+----------Doubly Stack System---------
| 0 |quit
+--------------------------------------
| 1 |initialize the stack
+--------------------------------------
| 2 |push elem
+--------------------------------------
| 3 |pop elem
+--------------------------------------
| 4 |get top
+--------------------------------------
| 5 |get all
+--------------------------------------
| 6 |get size
+--------------------------------------
input your choice: 1
input the size of the system: 100
----------INITIALIZATION FINISH----------
请按任意键继续. . .
```



选择栈功能测试如下：

```
+----------Doubly Stack System---------
| 0 |quit
+--------------------------------------
| 1 |initialize the stack
+--------------------------------------
| 2 |push elem
+--------------------------------------
| 3 |pop elem
+--------------------------------------
| 4 |get top
+--------------------------------------
| 5 |get all
+--------------------------------------
| 6 |get size
+--------------------------------------
input your choice: 2
input the number you want to push: -1
input a id(0 for the 1st, 1 for the 2nd): -1
illegal ID, input again:
```

1.2 团队队列类模板

1.2.1 实验内容

有 n 个团队的人进行排队。在团队队列中每个元素属于一个团队，同一团队中的元素在队列中的位置是连续的。当一个元素入队时，如果队列中已有同团队元素，则该元素入队并排在同团队的最后。否则，该元素入队并放在整个大团队最后。出队操作同队列一样，大团队队首元素出队。

1.2.2 主要算法设计

既然是队列类模板，那就要尽量保证团队队列模板仍然保留队列的优秀性质，即出队与入队的时间复杂度均为$\boldsymbol{O}_{(1)}$。注意到每一个元素所属团队是给定的，且属于 1 到 n。考虑对每一个团队开一个队列，然后再整体开一个大队列。用一个 siz 数组记录每一个小团队队列中元素个数。插入元素时，先查询小团队是否为空：如果为空，则先在大队列中插入小团队的团队编号，再将该元素插入所属小团队的队列末尾，然后更新 siz 数组；如果非空，则直接在所属小团队的队列末尾插入该元素，并更新 siz 数组。队首出队时，先取出大队列的队首元素，也就是出队元素所属小团队的编号，然后在所属小团队的队列里

弹出队首，更新 siz 数组。更新后如果恰好 siz 值变为 0，则在大团队队列里弹出队首。

1.2.3 主要数据组织

完成了 Node 类与 GroupQueue 的设计，并且使用了 LinkQueue 类模板。

对于 Node 类而言，其中有数据成员：

```
ElemType data;                  // 数据域
Node<ElemType> *next;           // 指针域
```

成员函数：

```
Node();                         // 无参数的构造函数
Node(ElemType item, Node<ElemType> *link = NULL);  // 已知数数据元素值和指针建立结构
```

表 2. GroupQueue **类中数据成员与函数的声明**

数据成员 / 函数原型	返回值类型	功能
Node();		无参数的构造函数
Node(ElemType item, Node<ElemType> *link;		构造函数已知数据元素值和指针建立结构
int n,s;		N 为团队个数，在构造时确定 s 表示队列总长度
int siz[1010];		各团队在队列中的数量
LinkQueue<T> q[1010];		对各种团队都各开一个队列
LinkQueue<int> que;		对总的团队开个队列
Status DeleteElem(int i, T &e);	Status	删除指定元素
Status InsertElem(int i, const T &e);	Status	插入指定元素

1.2.4 测试分析

以下测试了团队的入队与出队操作，其中，使用了队列为空的边界条件，确保程序的正确。

```
开始测试!
2
执行出队操作。
队列为空，出队失败。
1 1 1
执行入队操作，入队元素为1,它属于1团队。
1 1 2
执行入队操作，入队元素为1,它属于2团队。
1 2 1
执行入队操作，入队元素为2,它属于1团队。
1 3 3
执行入队操作，入队元素为3,它属于3团队。
2
执行出队操作。
出队的元素是1 ,它属于1团队。
2
执行出队操作。
出队的元素是2 ,它属于1团队。
2
执行出队操作。
出队的元素是1 ,它属于2团队。
2
执行出队操作。
出队的元素是3 ,它属于3团队。
```

2 综合性实验

2.1银行排队系统

2.1.1实验内容

银行排队模拟系统，实现三个服务窗口，对标三种客户。功能包括：取号、排队、服务、管理等。

2.1.2 算法分析

模拟银行排队，本质上还是队列。因为有三种客户，所以开三个队列来代表三种不同的服务窗口。当一个客户来取号排队时，根据他的类型，决定把他插入哪一个队列里。至于窗口业务办理情况，本算法以当前最新取号的人进入大厅的时间为标准，将其作为现在的时间，然后分别遍历三个窗口的队列，将已经完成业务的客户出队。

2.1.3 主要数据组织

Client 类有数据成员：

int number, hour, minute, second, type, time, delay;

Client 是客户类，number 是取号编号，hour、minute、second 则是客户进入银行的时间，type 是客户类型，time 是客户所需要的服务时间，delay 是客户需要等多久

Queue 类有数据成员：

int length, tot, solved, waittime, solvetime, resttime, pretime;

Node *head,*rear;

其中 length 是队列中元素个数，tot 是当前窗口排号进行到的编号，solved 是该窗口已经完成的业务数，resttime 是窗口休息时间（这段时间内没有人需要服务），waittime 是这个窗口客户排队总时长，solvetime 是业务处理时间，pretime 是这个队列上一个出队的客人的离开时间。

表 3. Queue 类中数据成员与函数的声明

函数原型	返回值类型	功能
void Pop();	void	队首出队
void Push(const Client &x);	void	元素入队
bool IsEmpty();	bool	队列是否为空
int GetWaitTime();	int	返回 resttime
int GetSolved();	int	返回 solved 的业务数
int GetSolveTime();	int	返回 solvetime
int GetTot();	int	返回 tot
int GetRestTime();	int	返回 resttime
int GetPreTime();	int	返回 pretime
void SetPreTime(int Pretime=0);	void	设置 pretime
void AddWaitTime(int Add);	void	修改 waittime，增加 Add
void AddRestTime(int Add);	void	修改 resttime，增加 Add
Node* GetHead();	Node*	返回队首
Node* GetRear();	Node*	返回队尾

2.1.4 数据测试

一组简单的测试，0:0:0 进来一个客户属于第一类，得到排号 A1，且需要 30 分钟服务时间，则 0:0:0 到 0:0:30 这段时间一号窗口均在处理这一个客户。0:0:20 进来一个客户属于 C 类，得到排号 C1，需要 50 分钟服务时间，则 0:0:20 到 0:1:10 这段时间三号窗口服务该客户。0:0:10 的时候进来一个 A 类客户，

他在一号窗口排队，排号 A2，需要等待时间是 20s，因为目前 A1 刚进行了 10s 服务。

```
请选择操作：
 0：退出 1:取号 2:查询业务数 3：查询等待时间 4：查询业务办理时间
1
请输入客户类型（1：企业客户 2：VIP客户 3：普通客户）
1
请输入客户到达时间：(时 分 秒)
0 0 0
请输入客户需要的服务时长：
30
获得编号：A1
完成取号
请选择操作：
 0：退出 1:取号 2:查询业务数 3：查询等待时间 4：查询业务办理时间
1
请输入客户类型（1：企业客户 2：VIP客户 3：普通客户）
3
请输入客户到达时间：(时 分 秒)
0 0 20
请输入客户需要的服务时长：
50
获得编号：C1
完成取号
请选择操作：
 0：退出 1:取号 2:查询业务数 3：查询等待时间 4：查询业务办理时间
1
请输入客户类型（1：企业客户 2：VIP客户 3：普通客户）
1
请输入客户到达时间：(时 分 秒)
0 0 10
请输入客户需要的服务时长：
20
获得编号：A2
完成取号
请选择操作：
 0：退出 1:取号 2:查询业务数 3：查询等待时间 4：查询业务办理时间
3
在企业窗口的等待时间：20
在VIP窗口的等待时间：0
在普通窗口的等待时间：0
请选择操作：
 0：退出 1:取号 2:查询业务数 3：查询等待时间 4：查询业务办理时间
_
```

3.1 课程设计中遇到的问题和解决方法

在设计团队队列时，经过了小组讨论与搜集资料，我们对算法的时间复杂度进行了优化，特殊位置、临界位置进行了特殊处理，保证了程序的健壮性。

3.2 实验总结

验证性实验是有助于夯实基础，对于综合性实验，我们不仅仅应当注意是否能够解决问题，更要注重优化算法的时间复杂度。严谨性也同样重要，代码实现以后，一定要通过多组数据检验，确保其正确性。