项目说明文档

数据结构课程设计

——银行业务

作 者 姓 名: 沈星宇

学　　号: 1951576

指 导 教 师: 张颖

学院、 专业: 软件学院 软件工程

同济大学

Tongji University

一、 分析

（1） 应用背景

设某银行有 A，B 两个业务窗口，且处理业务的速度不一样，其中 A 窗口处理速度是 B 窗口的 2 倍----即当 A 窗口每处理完 2 个顾客时，B 窗口处理完 1 个顾客。给定到达银行的顾客序列，请按照业务完成的顺序输出顾客序列。假定不考虑顾客信后到达的时间间隔，并且当不同窗口同时处理完 2 个顾客时，A 窗口的顾客优先输出。

（2） 项目功能要求

1、输入说明：输入为一行正整数，其中第一数字 N（N<=1000）为顾客总数，后面跟着 N 位顾客的编号。编号为奇数的顾客需要到 A 窗口办理业务，为偶数的顾客则去 B 窗口。数字间以空格分隔。

2、输出说明：按照业务处理完成的顺序输出顾客的编号。数字键以空格分隔，但是最后一个编号不能有多余的空格。

3、测试用例：

序号	输入	输出	说明
1	8 2 1 3 9 4 11 13 15	1 3 2 9 11 4 13 15	正常测试，A 窗口人多
2	8 2 1 3 9 4 11 12 16	1 3 2 9 11 4 12 16	正常测试，B 窗口人多
3	1 6	6	最小 N

二、 设计

（1） 数据结构设计

```
class ListPoint {
private:
    int data;
    ListPoint* left;
    ListPoint* right;
    friend class List;
public:
    ListPoint(const int& item);
    bool insert(ListPoint*& curPosition,ListPoint*& newNode);
    int getNum();
};
```

```
class List {
private:
    ListPoint* head;
    ListPoint* tail;
    int length;
    friend class Queue;
public:
    List();
    bool makeEmpty();
    bool reset();
    int getLength();
    ListPoint* find(int location);
    bool insert(int value, int location);
    bool remove(int location);
    bool check(int value);
};
```

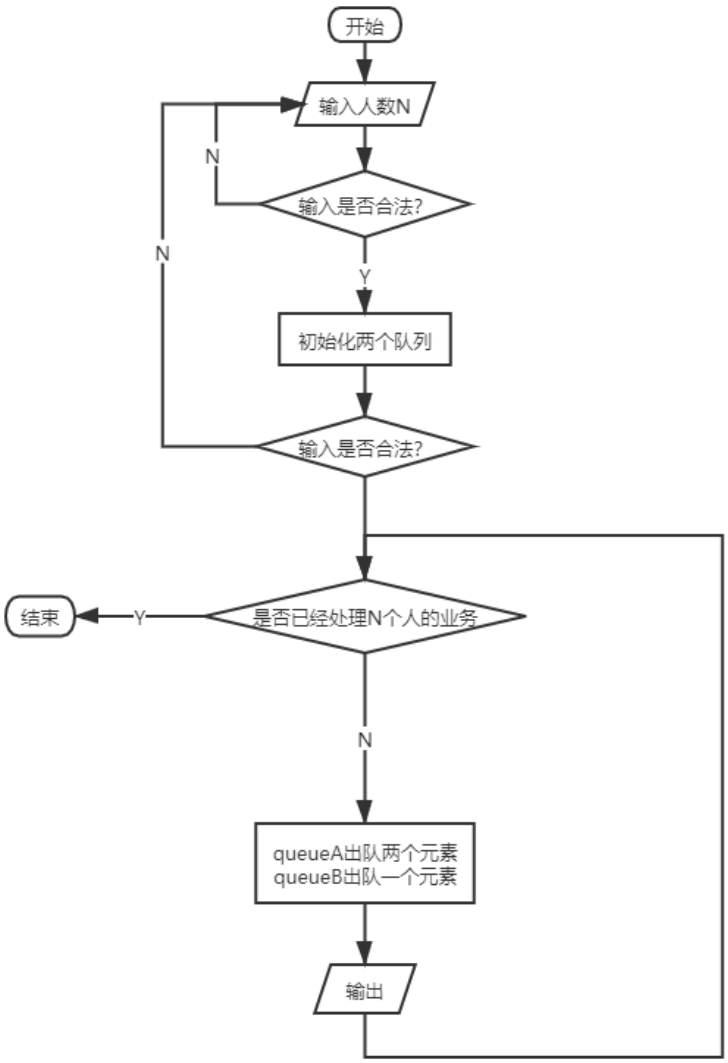
```
class Queue {
private:
    List list;
public:
    Queue();
    ~Queue();
    bool isEmpty();
    bool push(int value);
    bool pop(int& value);
    void reset();
    bool check(int value);
};
```

1、ListPoint 是结点的结构，内部包含了存储的数字 data 和指针 left、right，为了锻炼写双链表的能力才写成了双链表，其实本项目单链表即可实现

2、List 是链表的结构，里面包含了常规的操作的函数，为了使 List 能够访问 ListPoint 的成员，故将其在 ListPoint 类里写作了友元。

3、Queue 是队列的结构，里面包含了一个链表（链式的队列），为了使 Queue 能够访问 List 的成员，故将其再 List 类里面写作了友元。

（2） 程序流程设计



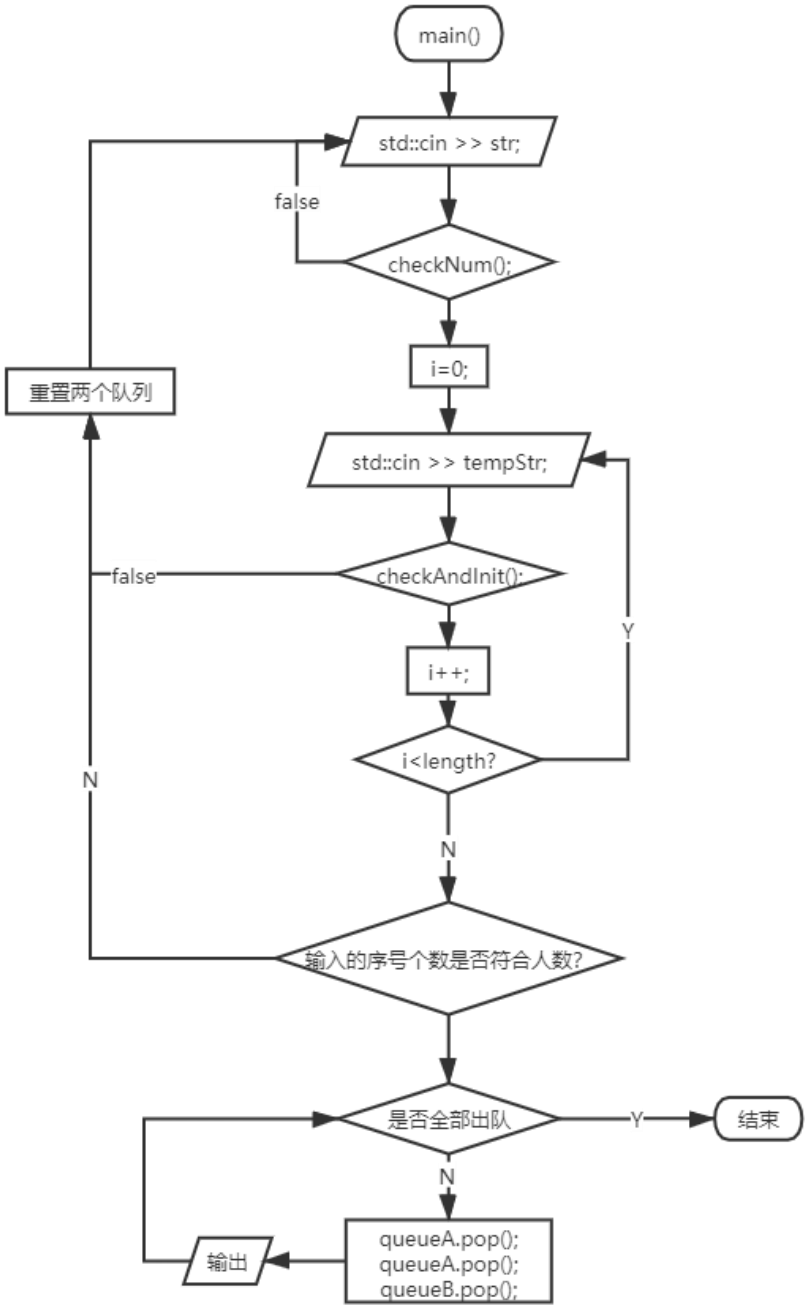
1、 首先输入的是一串数字序列，但我们先对第一个数字，即人数 N 进行合法性检测，非正整数输入均会被要求重新输入

2、 然后循环读入 N 个数字，即序号，注意第 N 个之后的数字不会被读入。读入数字的同时会根据它的奇偶性分别压入 queueA 或 queueB，同时会对数字进行合法性检测，如果读到非正整数或者读到重复的号码，都会被要求重新输入

3、 循环让队列中的元素出队，每次循环 queueA 出队两个元素，queueB 出队一个元素，一边出队一边输出

三、 实现

1、 main()函数的内部逻辑



（1） main()函数会首先读入第一个数字 str（人数），并调用 checkNum();函数对读入的字符串进行合法性检测，还通过传入 int& length 的形式将转化成 int 类型的人数传出。如果输入不合法则会要求重新输入。

（2） 接着连续读入 length 个数字，并调用 checkAndInit();函数对读入的字符串进行合法性检测（包括不能有重复的数字出现），并根据奇偶将其分别压入队列 queueA 和 queueB 当中。如果输入不合法则会要求重新输入。

（3） 结束输入后循环让队列中的元素出队，每次循环 queueA 出队两个元素，queueB 出队一个元素，一边出队一边输出

```
//---------------输入--------------------------
while (1) {
    std::cout << "-------------------------------------\n";
    std::cout << "请输入一行若干个正整数构成的序列\n";
    std::string str;
    std::cin >> str;
    if (!checkNum(str, length)) {
        continue;
    }
    //-----------------------------------------
    for (int i = 0; i < length; i++) {
        std::string tempStr;
        std::cin >> tempStr;
        if (!checkAndInit(tempStr, queueA, queueB)) {
            break;
        }
        count++;
    }
    if (count != length) {
        queueA.reset();
        queueB.reset();
    }
    else {
        break;
    }
}
```

```
int temp = 0;
count = 0;
while (count != length) {
    if (queueA.pop(temp)) {
        count++;
        std::cout << temp;
        if (count != length) {
            std::cout << ' ';
        }
    }
    if (queueA.pop(temp)) {
        count++;
        std::cout << temp;
        if (count != length) {
            std::cout << ' ';
        }
    }
    if (queueB.pop(temp)) {
        count++;
        std::cout << temp;
        if (count != length) {
            std::cout << ' ';
        }
    }
}
```

2、 checkNum()的实现

```
bool checkNum(const std::string& str, int& length) {
    for (int position = 0; str[position] != '\0'; position++) {
        if (str[position] >= '0' && str[position] <= '9') {
            length = length * 10 + str[position] - '0';
        }
        else {
            std::cout << "输入不合法，请输入人数 0 < N <= 1000.\n";
            return false;
        }
    }
    if (length > 1000 || length == 0) {
        return false;
    }
    return true;
}
```

(1) 对于输入的字符串，进行逐位的判断和操作，用 str[position] >= '0' && str[position] <= '9'来判断是否输入了非法字符，将其转化为 int 类型的 length

(2) 判断输入的数字是否超过 1000

3、 checkAndInit()的实现

```
bool checkAndInit(const std::string& str, Queue& queueA, Queue& queueB) {
    int temp = 0;
    for (int pos = 0; str[pos] != '\0'; pos++) {
        if (str[pos] > '0' && str[pos] <= '9') {
            temp = temp * 10 + str[pos] - '0';
        }
        else {
            std::cout << "输入不合法，正整数号码.\n";
            return false;
        }
    }
    if (temp % 2 == 0) {
        if (queueB.check(temp)) {
            queueB.push(temp);
        }
        else {
            return false;
        }
    }
    else {
        if (queueA.check(temp)) {
            queueA.push(temp);
        }
        else {
            return false;
        }
    }
    return true;
}
```

(1) 对于输入的字符串，进行逐位的判断和操作，用 str[position] >= '0' && str[position] <= '9'来判断是否输入了非法字符，将其转化为 int 类型的 temp

(2) 调用 Queue::check();来查看 temp 是否在前面的序列种已经出现过，若已经出现则说明该序列不合法，直接中止要求重新输入。根据 temp 的奇偶将其用 Queue::push();函数压入队列。

4、 Queue::push();和 Queue::pop();的实现

```
bool Queue::push(int value) {
    if (list.insert(value, list.getLength())) {
        return true;
    }
    else {
        return false;
    }
}
```

```
bool Queue::pop(int& value) {
    if (!isEmpty()) {
        value=list.find(1)->getNum();
        list.remove(1);
        return true;
    }
    else {
        return false;
    }
}
```

由于是链式的队列，所以 push()操作就是在链表的末尾插入元素，pop()操作就是将链表的首元素给删除。

5、 List::insert()的实现

insert()函数其实是按位后插，即在 location 的后面插入一个结点。

（1） 首先会对 location 的合法性进行检测（虽然本项目按照顺序插入在队尾不会涉及到这个内容，但是为了安全性和整体性仍然进行检测）。禁止在第 0 位及第 0 位之前插入结点；在超出链表的最后一个结点的时候在最后一个结点插入。

（2）然后调用 find()函数去查找到 location 位置的结点，用 malloc 为新的结点动态分配一个内存，修改新节点 newNode 的 data 并且调用 newNode 的 insert()函数将它的指针链接上去。

（3）调整尾指针，使链表的长度增加。

```
bool List::insert(int value, int location) {
    if (location < 0) {
        return false;
    }
    else if (location > length) {
        location = length;
    }

    ListPoint* p = head;
    p = find(location);  //后插操作要找到location的前一个结点

    ListPoint* newNode = (ListPoint*)malloc(sizeof(ListPoint));
    newNode->data = value;
    if (!newNode->insert(p, newNode)) {
        return false;
    }
    if (newNode->right == NULL) {
        tail = newNode;
    }
    length++;
    return true;
}
```

6、 List::find()的实现

按位查找，用一个指针从第一个结点指向最后一个结点，该函数返回所寻找位置的结点指针。

```
ListPoint* List::find(int location) {
    ListPoint* p = head;
    int count = 0;
    while (1) {
        if (p == NULL || count == location) {
            break;
        }
        p = p->right;
        count++;
    }
    return p;
}
```

7、 **ListPoint::insert()的实现**

后插操作要不存在 curPositon==NULL 的情况，即不在空指针后面插入内容。把结点用指针链接起来。因为最后一个结点的 right 指针指向的是 NULL，如果是在链表的末尾插入结点，那么不需要将 newNode 之后的结点（其实是空指针）的 left 指针指向 newNode，因为空指针不存在 left 指针。

```
bool ListPoint::insert(ListPoint*& curPosition, ListPoint*& newNode) {
    if (curPosition == NULL) {          //后插操作，所以不存在curPosition==NULL的插入
        return false;
    }
    newNode->left = curPosition;
    newNode->right = curPosition->right;
    if (curPosition->right != NULL) {   //在末尾插入就不需要后面的NULL指针指向newNode了
        newNode->right->left = newNode;
    }
    curPosition->right = newNode;
    return true;
}
```

8、 **List::reset()和 List::makeEmpty()的实现**

（1）在程序接收到不合法的输入时，程序会将链表清空，并且请求用户重新输入数字序列，直到输入的序列合法为止。实现这一部分功能的是 reset()和 makeEmpty()函数。

（2）makeEmpty()是释放内存的作用，在 main()函数的末尾、程序结束时也会被调用，防止碎片化内存块的出现。它通过指针 p 去访问链表的每一个结点，将每一个结点的内存释放，然后重置链表的长度 length = 0。

（3）对于 reset()函数，用于清空重置链表，内部会先调用 makeEmpty()释放内存，然后将头指针的 right 指针指向 NULL（防止访问脏数据），将尾指针指向头结点。

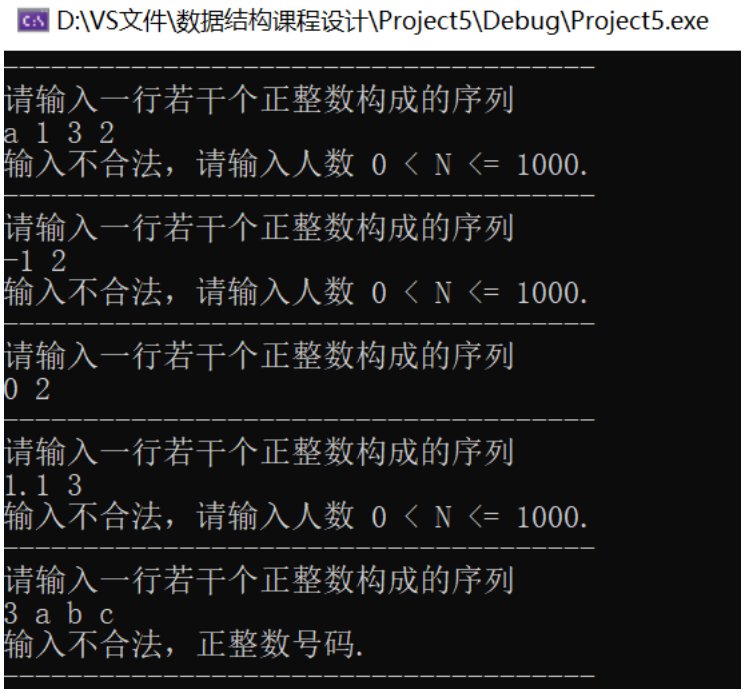
```
bool List::makeEmpty() {
    ListPoint* p = head->right;
    ListPoint* freePoint = p;
    while (1) {
        if (p == NULL) {
            length = 0;
            return true;
        }
        freePoint = p;
        p = p->right;
        free(freePoint);
    }
}

bool List::reset() {
    if (!makeEmpty()) {
        return false;
    }
    head->right = NULL;
    tail = head;
    return true;
}
```

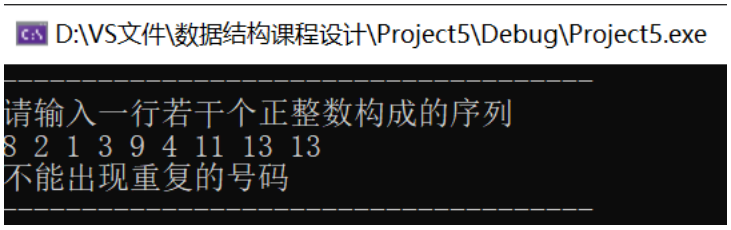
四、 测试

1、 初始化表格的合法性检测

（1） 输入非正整数、浮点数和字符要求重新输入



2、不允许出现重复的号码



3、 一般情况