



中国科学技术大学  
University of Science and Technology of China

## 数据结构实验报告 2

### 栈与队列的应用

姓名：\_\_\_\_\_高茂航\_\_\_\_\_

学号：\_\_\_\_\_PB22061161\_\_\_\_\_

日期：\_\_\_\_\_2023 年 10 月 30 日\_\_\_\_\_

# 实验报告 2

## 1 括号匹配检验

### 1.1 问题描述

假设一个表达式有英文字母(大、小写)、数字、四则运算符(+, -, \*, /)和左右小括号、中括号、大括号构成,以“@”作为表达式的结束符。请编写一个程序检查表达式中的左右大中括号是否匹配,若匹配,则返回“Yes”;否则返回“No”。

### 1.2 算法描述

#### 1.2.1 数据结构

用链栈来储存尚未完成匹配的括号。

```
1  typedef struct StackList{
2      char ope; //储存括号的链栈节点
3      struct StackList* next;
4  }Stack;
5  Stack* PushStack(Stack** S, char c){//将括号进栈
6      Stack *p=(Stack*)malloc(sizeof(Stack));
7      p->ope=c;
8      if(!*S){
9          *S=p;
10         (*S)->next=NULL;
11     }
12     else{
13         p->next=*S;
14         *S=p;
15     }
16     return *S;
17 }
18 char PopStack(Stack** S){//将栈顶括号出栈
19     char c;
20     Stack *p=*S;
21     c=p->ope;
22     (*S)=(*S)->next;
23     free(p);
24     return c;
25 }
26 char GetStackTop(Stack* S){//取栈顶括号
27     if(!S)
28         return 0;
29     return S->ope;
30 }
31
```

Listing 1: 链栈相关操作

#### 1.2.2 括号匹配检验算法

每次读到左括号就进栈,读到右括号就出栈,如果出栈的括号与读到的右括号不匹配,则返回“No”,如果最后栈为空,则返回“Yes”。

```
1  int BracketTest(char* a){//检验括号是否匹配
2      int i=0;
3      Stack* S=NULL;
4      char x;
```

## 实验报告 2

```
5     while(a[i]&&a[i]!='@'){
6         switch (a[i]) {
7             case '(':
8             case '[':
9             case '{':
10            PushStack(&S,a[i]);
11            i++;
12            break;
13            case ')':{
14                if(!S)
15                    return 0;
16                char c1=PopStack(&S);
17                if(c1=='['||c1=='{')
18                    return 0;
19                i++;
20                break;
21            }
22            case ']':{
23                if(!S)
24                    return 0;
25                char c2=PopStack(&S);
26                if(c2=='{'||c2=='('||(GetStackTop(S)&&GetStackTop(S)=='('))//判断优先级
27                    return 0;
28                i++;
29                break;
30            }
31            case '}':{
32                if(!S)
33                    return 0;
34                char c3=PopStack(&S);
35                if(c3=='('||c3=='['||(GetStackTop(S)&&(GetStackTop(S)=='('||GetStackTop(S)=='[')))
36                    return 0;
37                i++;
38                break;
39            }
40            default:{
41                i++;
42                break;
43            }
44        }
45    }
46    if(S)
47        return 0;//最后栈不为空，说明有左括号没有匹配到右括号
48    return 1;
49 }
50
```

Listing 2: 括号匹配检验算法

### 1.2.3 主函数(包括读取文件操作)

2.txt 文件内容为:

```
1 6
2 3*(4+5)-{6/[7*(8-9)]}@
3 {[1+2*(3-4)]/5}+6-7@
4 1+[2*(3-4)]/5@
5 6*{7+[8-(9*10)]}@
6 8/{9+(10-[11*12])}@
```

## 实验报告 2

```
7  {[(2+3*4)/(5-6)+7]}@
8
```

Listing 3: 2.txt

```
1  int main(){
2      int n,i;
3      FILE *fp=fopen("2.txt","r");
4      if (!fp) {
5          fprintf(stderr, "无法打开文件\n");
6          return 0;
7      }
8      fscanf(fp, "%d", &n);
9      fgetc(fp); //读掉换行符
10     char st[10][50];
11     for(i=0;i<n;i++){
12         fgets(st[i],sizeof(st[i]),fp); //st[i]是第i个字符串
13     }
14     for(i=0;i<n;i++){
15         if(BracketTest(st[i]))
16             cout<<"Yes"<<endl;
17         else
18             cout<<"No"<<endl;
19     }
20     fclose(fp);
21 }
```

Listing 4: 主函数

### 1.3 调试分析

1. 进栈出栈函数形参没有设置为指向指针的指针以达到修改指针指向的目的;
2. 忽略了完成匹配后栈不为空的情况。

### 1.4 算法时空分析

由于采用栈的链式存储,所以空间复杂度和时间复杂度都为  $O(n)$ 。

### 1.5 测试结果



```
F:\Desktop\code\Data Structure\Stack_Queue\BracketTest.exe
Yes
Yes
No
No
Yes
No

-----
Process exited after 0.09614 seconds with return value 0
请按任意键继续...
```

# 实验报告 2

## 2 银行业务模拟

### 2.1 问题描述

银行客户业务分为两种: 第一种是申请从银行得到一笔资金, 即取款或借款; 第二种是向银行投入一笔资金, 即存款或还款。

银行有两个服务窗口, 相应地有两个队列。客户到达银行后先排第一个队, 处理每个客户业务时, 如果属于第一种, 且申请额超出银行现存资金总额而得不到满足, 则立刻排入第二个队等候直至满足时才离开银行; 否则业务处理完后立刻离开银行, 每接待完一个第二种业务的客户, 则顺序检查和处理 (如果可能) 第二个队列中的客户, 对能满足的申请者予以满足, 不能满足者重新排到第二个队列的队尾。注意, 在此检查过程中, 一旦银行资金总额少于或等于刚才第一个队列中最后一个客户 (第二种业务) 被接待之前的数额, 或者本次已将第二个队列 检查或处理了一遍, 就停止检查 (因为此时已不可能还有能满足者), 转而继续接待第一个队列的客户。任何时刻都只开一个窗口。假设检查不需要时间, 营业时间结束时所有客户立即离开银行。写一个上述银行业务的事件驱动模拟系统, 通过模拟方法求出客户在银行内逗留的平均时间。

### 2.2 算法描述

#### 2.2.1 数据结构

```
1  typedef struct QueueList{
2      int num; //顾客序号
3      struct QueueList* next;
4  }Queue;
5  Queue* EnQueue(Queue** Q,int n){//将顾客进队
6      Queue *p=(Queue*)malloc(sizeof(Queue));
7      p->num=n;
8      if(!*Q){
9          *Q=p;
10         (*Q)->next=NULL;
11     }
12     else{
13         Queue* q=*Q;
14         while(q->next)
15             q=q->next;
16         q->next=p;
17         p->next=NULL;
18     }
19     return *Q;
20 }
21 int DeQueue(Queue** Q){//将队首顾客出队
22     if(!*Q)
23         return -1;
24     int i;
25     Queue *p=*Q;
26     i=p->num;
27     *Q=(*Q)->next;
28     free(p);
29     return i;
30 }
31
```

Listing 5: 链队列基本操作

## 实验报告 2

### 2.2.2 模拟过程

```
1  int main(){
2      int n,i,arrive[50]={0},status[50],Time=0,wait[50]={0},waittotal=0,waitaverage=0,
        closetime=0,avetime=0,Money=0,money[50]={0},moneytemp=0;
3      //n为顾客人数, arrive数组记录每个顾客来到银行的时间, status数组记录每个顾客的状态, -1为
        没来银行, 0为正在等待, 1为已离开
4      //Time表示当前时间, wait数组记录每个顾客等待的时间, waittotal是所有顾客等待的总时间,
        waitaverage是顾客等待的平均时间, closetime是银行关门时间, avetime是每笔交易的平均时间
5      //Money为银行一开始的款额, money数组记录每个顾客的交易金额, 正数为存钱, 负数为取钱,
        moneytemp用来记录第一个队列最后一个客户(第二种业务)接待前银行的剩余款额
6      cin>>n>>Money>>closetime>>avetime;
7      Queue* Q=NULL;
8      for(i=0;i<n;i++){
9          cin>>money[i]>>arrive[i];
10         status[i]=-1;
11     }
12     for(i=0;i<n;i++){
13         if (arrive[i]>=closetime-avetime) {//在关门前完成不了交易相当于一直等到关门
14             status[i]=0;
15             break;
16         }
17         else {
18             if(Time<arrive[i])
19                 Time=arrive[i]; //重置当前时间
20             if(money[i]<0){ //第一种业务: 取钱
21                 if(-money[i]<=Money){ //银行有足够余额
22                     Money+=money[i];
23                     wait[i]=Time-arrive[i];
24                     status[i]=1;
25                     Time+=avetime; //重置当前时间
26                     moneytemp=Money;
27                     if(Time>=closetime)
28                         loop:break;
29                 }
30                 else{ //银行没有足够余额, 进入第二条队等待
31                     status[i]=0;
32                     EnQueue(&Q,i);
33                 }
34             }
35             else{ //第二种业务
36                 Money+=money[i];
37                 wait[i]=Time-arrive[i];
38                 status[i]=1;
39                 Time+=avetime; //重置当前时间
40                 int status2[50]={0}; //用来标记第二条队的顾客是否被检查到, 没检查是0, 检查到是1
41                 while (Money>moneytemp){
42                     int k=DeQueue(&Q);
43                     if(k==-1) break;
44                     else if(!status2[k]){
45                         if(-money[k]<=Money){
46                             Money+=money[k];
47                             wait[k]=Time-arrive[k];
48                             status[k]=1;
49                             Time+=avetime; //重置当前时间
50                             moneytemp=Money;
51                             if(Time>=closetime)
52                                 goto loop; //跳出while循环和for循环
53                         }
54                     }
55                 }
56             }
57         }
58     }
59 }
```

## 实验报告 2

```
54         else{
55             status2[i]=1;
56             EnQueue(&Q,i);//不满足就重新入队
57         }
58     }
59 }
60 }
61 }
62 }
63 for ( i = 0; i < n; i++){
64     if(!status[i])
65         wait[i]=closetime-arrive[i];
66     cout<<"第"<<i+1<<"个顾客的等待时间为"<<wait[i]<<endl;
67 }
68 for ( i = 0; i < n; i++)
69     waittotal+=wait[i];
70 waitaverage=waittotal/n;
71 cout<<n<<"个顾客的平均等待时间为"<<waitaverage<<endl;
72 }
73 }
```

Listing 6: 模拟过程

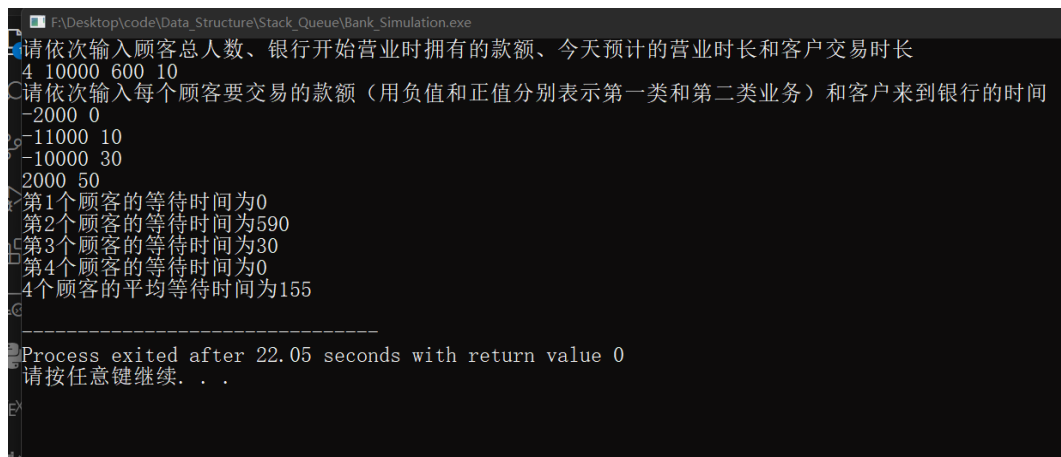
### 2.3 调试分析

出现问题不多,关键在于理清事件发生的逻辑,并注意时间的标记。

### 2.4 算法时空分析

由于采用队列的链式存储,所以空间复杂度和时间复杂度都为  $O(n)$ 。

### 2.5 测试结果分析



```
F:\Desktop\code\Data_Structure\Stack_Queue\Bank_Simulation.exe
请依次输入顾客总人数、银行开始营业时拥有的款额、今天预计的营业时长和客户交易时长
4 10000 600 10
请依次输入每个顾客要交易的款额（用负值和正值分别表示第一类和第二类业务）和客户来到银行的时间
-2000 0
-11000 10
-10000 30
2000 50
第1个顾客的等待时间为0
第2个顾客的等待时间为590
第3个顾客的等待时间为30
第4个顾客的等待时间为0
4个顾客的平均等待时间为155

-----
Process exited after 22.05 seconds with return value 0
请按任意键继续. . .
```

1 号顾客 0 时刻到达银行,于 10 时办理完手续离开,此时银行有款额 8000 元; 2 号顾客 10 时刻到达银行,此时银行内存款不足,进入队列 2 ; 3 号顾客 30 时刻到达银行,此时银行内存款不足,进入队列 2 ; 4 号顾客 50 时刻到达银行,于 60 时办理完手续离开,此时银行有款额 10000 元。这时扫描队列 2 ,队列 2 中第一个顾客 2 号顾客的要求仍然不满足, 2 号顾客出队再入队; 此时队列 2 中第一个顾客 3 号客户满足要求,因此 3 号顾客于 60 时刻开

## 实验报告 2

---

始办理手续,于 70 时刻办理完毕离开银行,等待时间为  $60 - 30 = 30$ ,此时银行有余款 0 元。然后直到银行关门时间 600 时刻 2 号客户都没有离开银行,等待时间为  $600 - 10 = 590$ 。

### 3 实验体会收获

加深了对栈和队列结构的理解,也对栈和队列的相关操作更加熟悉,并对栈和队列的应用有了更深的认识,此外也复习了读取文件的基本操作。