

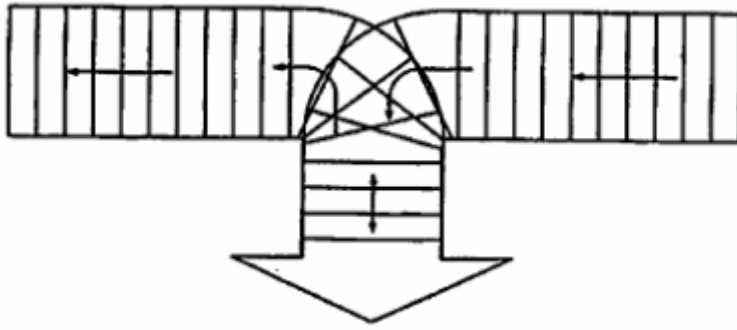
### 3.3.1

假设一个算术表达式中包含圆括号、方括号和花括号 3 种类型的括号，编写一个算法来判别表达式中的括号是否配对，以字符 '\0' 作为算术表达式的结束符。

```
1 //参考 栈在括号匹配中的应用.cpp
2 bool bracketCheck(string str,int len)
3 {
4     SqStack S;
5     InitStack(S);
6
7     for(int i=0;i<len;i++)
8     {
9         if(str[i]=='('||str[i]=='['||str[i]=='{')
10             Push(S,str[i]);
11         else
12         {
13             if(StackEmpty(S)==true)//栈空(右括号无匹配)
14                 return false;
15             ElementType x;
16             Pop(S,x);
17
18             if(str[i]==')'&&x!='(')//左右括号不匹配
19                 return false;
20             if(str[i]==']'&&x!='[')
21                 return false;
22             if(str[i]=='}'&&x!='{')
23                 return false;
24         }
25     }
26     return StackEmpty(S);//左括号不匹配
27 }
28
29 if(bracketCheck(s,s.size())==true)
30     cout << "匹配成功" << endl;
31 else cout << "匹配失败" << endl;
```

### 3.3.2

按下图所示铁道进行车厢调度（注意，两侧铁道均为单向行驶道，火车调度站有一个用于调度的“栈道”），火车调度站的入口处有  $n$  节硬座和软座车厢（分别用  $H$  和  $S$  表示）等待调度，试编写算法，输出对这  $n$  节车厢进行调度的操作（即入栈或出栈操作）序列，以使所有的软座车厢都被调整到硬座车厢之前。



设车道为  $B$ , 每次将硬座  $H$  入栈, 软座  $S$  入车道  $B$  中, 当所有的座位都检查完, 将硬座  $H$  出栈接在车道  $B$  之后, 即可实现所有的软座车厢都被调整到硬座车厢之前.

```

1  string trainArrange(string str,int len)
2  {
3      SqStack S;
4      InitStack(S);
5      string res="";
6
7      for(int i=0;i<len;i++)
8      {
9          if(str[i]=='S')//软座
10             res+='S';
11         else //硬座
12             Push(S, str[i]);
13     }
14
15     while(StackEmpty(S)==false)//将剩余硬座接在软座之后
16     {
17         char seat;
18         Pop(S, seat);
19         res+=seat;
20     }
21     return res;
22 }

```

### 3.3.3

利用一个栈实现以下递归函数的非递归计算:

$$P_n(x) = \begin{cases} 1, n = 0 \\ 2x, n = 1 \\ 2xP_{n-1}(x) - 2(n-1)P_{n-2}(x), n > 1 \end{cases}$$

```

1 //递归实现
2 int P(int n,int x)
3 {
4     if(n==0)
5         return 1;
6     else if(n==1)
7         return 2*x;
8     else return 2*x*P(n-1,x)-2*(n-1)*P(n-2,x);
9 }

```

已知

- $P_0(x) = 1;$
- $P_1(x) = 2x;$
- $P_2(x) = 2xP_1(x) - 2 \times 1P_0(x);$
- $P_3(x) = 2xP_2(x) - 2 \times 2P_1(x);$
- $\dots;$
- $P_{n-1}(x) = 2xP_{n-2}(x) - 2 \times (n-2)P_{n-3}(x);$
- $P_n(x) = 2xP_{n-1}(x) - 2 \times (n-1)P_{n-2}(x).$

若要求出  $P_n(x)$ , 需求出  $P_{n-1}(x), P_{n-2}(x), \dots, P_1(x), P_0(x)$ .

设  $num0 = P_0(x), num1 = P_1(x)$ , 此时  $P_2(x) = 2x \times num1 - 2 \times num0$ ;

同理若想求出  $P_3(x)$ , 此时让  $num0 = num1, num1 = P_2(x)$ .

最终求出  $P_n(x)$  后,  $num1$  为最终的答案.

不用像书上那样存入 结构体 中, 只用记录其对应的下标  $id$  即可.

```

1 int P(int n,int x)
2 {
3     if(n==0)
4         return 1;
5
6     SqStack S;
7     InitStack(S);
8
9     int num0=1,num1=2*x;
10
11     for(int i=n;i>=2;i--)//i为对应的下标id
12         Push(S,i);
13
14     while(StackEmpty(S)==false)
15     {
16         int id;
17         Pop(S, id);
18         int val=2*x*num1-2*(id-1)*num0;
19
20         num0=num1;

```

```

21     num1=val;
22 }
23 return num1;
24 }

```

### 3.3.4

某汽车轮渡口，过江渡船每次能载 10 辆车过江。过江车辆分为客车类和货车类，上渡船有如下规定：同类车先到先上船；客车先于货车上船，且每上 4 辆客车，才允许放上 1 辆货车；若等待客车不足 4 辆，则以货车代替；若无货车等待，允许客车都上船。试设计一个算法模拟渡口管理。

**模拟 队列** 操作(此处**不能用栈**，由于有先来先上船的条件)注意每条船最多容纳 10 辆车，此处考虑**一个一个到达**，设客车为 1，货车为 0，无论客车的数量多少，货车优先进入等待队列；若当前为客车，若当前船上的客车数量  $\geq 4$  时(有可能**客车到达的频率比货车早**)且货车等待队列不为空且船有足够空间，则让货车出等待队列，再处理当前的客车是否需要新开一辆，最后将剩余货车等待队列中的元素出队，放进船中(注意考虑船的容纳情况)——想的比较细，可能仍存在部分错误。

```

1 void ShipArrange(int n)
2 {
3     int ship_tot=0,bus_tot=0,van_tot=0; //分别表示船上车的数量,客车数量,货车队列中的数量
4     int ship_cnt=0; //船的个数
5
6     SQueue van_Q; //存储货车的序号
7     SQueue ship_Q; //存储当前船上的情况
8
9     InitQueue(van_Q);
10    InitQueue(ship_Q);
11
12    for(int i=1;i<=n;i++)
13    {
14        if(ship_tot<=9) //船有足够的空间(小于10)
15        {
16            if(a[i]==1) //客车
17            {
18                if(bus_tot>=4&&QueueEmpty(van_Q)==false) //货车队列有且当前客车数量
                不小于3
19                {
20                    int num;
21                    DeQueue(van_Q, num); //出货车队
22                    van_tot--; //货车队列中的数量减少
23                    ship_tot++; //货车上船
24                    EnQueue(ship_Q, num);
25
26                    bus_tot=0;
27                }
28
29                if(ship_tot<=9) //由于有可能刚好把货车放入使得船满
30                {
31                    EnQueue(ship_Q, i);
32                    bus_tot++;

```

```

33         ship_tot++;
34     }
35     else if(ship_tot==10)
36     {
37         cout << "第" << ++ship_cnt << "辆船:";
38         PrintQueue(ship_Q);
39         ship_tot=0,bus_tot=0;
40         ClearQueue(ship_Q);
41
42         EnQueue(ship_Q, i);
43         bus_tot++;
44         ship_tot++;
45
46     }
47 }
48 else//货车
49 {
50     EnQueue(van_Q, i);
51     van_tot++;
52 }
53 }
54
55 if(ship_tot==10)
56 {
57     cout << "第" << ++ship_cnt << "辆船:";
58     PrintQueue(ship_Q);
59     ship_tot=0,bus_tot=0;
60     ClearQueue(ship_Q);
61 }
62
63 //cout << i << ": " << ship_tot << " " << bus_tot << " " << van_tot <<
endl;
64 }
65
66
67 while(QueueEmpty(van_Q)==false)
68 {
69     if(ship_tot<=9)
70     {
71         int num;
72         DeQueue(van_Q, num);//出货车队
73         van_tot--;//货车队列中的数量减少
74         ship_tot++;//货车上船
75         EnQueue(ship_Q, num);
76     }
77     else if(ship_tot==10)
78     {
79         cout << "第" << ++ship_cnt << "辆船:";
80         PrintQueue(ship_Q);
81         ship_tot=0;
82         ClearQueue(ship_Q);
83

```

```

84         int num;
85         DeQueue(van_Q, num); //出货车队
86         van_tot--; //货车队列中的数量减少
87         ship_tot++; //货车上船
88         EnQueue(ship_Q, num);
89     }
90
91     if(ship_tot==10) //此处注意细节
92     {
93         cout << "第" << ++ship_cnt << "辆船:";
94         PrintQueue(ship_Q);
95         ship_tot=0;
96         ClearQueue(ship_Q);
97     }
98 }
99
100
101 if(ship_tot<=9)
102 {
103     cout << "第" << ++ship_cnt << "辆船:";
104     PrintQueue(ship_Q);
105     ship_tot=0;
106     ClearQueue(ship_Q);
107 }
108 }

```