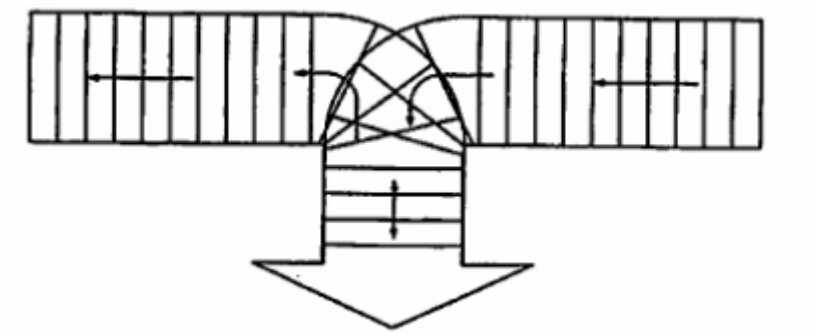
3.3.1

> 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.

```cpp
//参考 栈在括号匹配中的应用.cpp
bool bracketCheck(string str,int len)
{
    SqStack S;
    InitStack(S);

    for(int i=0;i<len;i++)
    {
        if(str[i]=='('||str[i]=='['||str[i]=='{')
            Push(S,str[i]);
        else
        {
            if(StackEmpty(S)==true)//栈空(右括号无匹配)
                return false;
            ElemType x;
            Pop(S,x);

            if(str[i]==')'&&x!='(')//左右括号不匹配
                return false;
            if(str[i]==']'&&x!='[')
                return false;
            if(str[i]=='}'&&x!='{')
                return false;
        }
    }
    return StackEmpty(S);//左括号不匹配
}

if(bracketCheck(s,s.size())==true)
    cout << "匹配成功" << endl;
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.

```
string trainArrange(string str,int len)
{
    SqStack S;
    InitStack(S);
    string res="";

    for(int i=0;i<len;i++)
    {
        if(str[i]=='S')//软座
            res+='S';
        else //硬座
            Push(S, str[i]);
    }

    while(StackEmpty(S)==false)//将剩余硬座接在软座之后
    {
        char seat;
        Pop(S, seat);
        res+=seat;
    }
    return res;
}
```

3.3.3

利用一个栈实现以下递归函数的非递归计算:

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

```
//递归实现
int P(int n,int x)
{
    if(n==0)
        return 1;
    else if(n==1)
        return 2*x;
    else return 2*x*P(n-1,x)-2*(n-1)*P(n-2,x);
}
```

已知

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

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

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.

```
int P(int n,int x)
{
    if(n==0)
        return 1;

    SqStack S;
    InitStack(S);

    int num0=1,num1=2*x;

    for(int i=n;i>=2;i--)//i为对应的下标id
        Push(S,i);

    while(StackEmpty(S)==false)
    {
        int id;
        Pop(S, id);
        int val=2*x*num1-2*(id-1)*num0;

        num0=num1;
```

```
        num1=val;
    }
    return num1;
}
```

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 ≥ 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.

```
void ShipArrange(int n)
{
    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
    int ship_cnt=0;//船的个数

    SqQueue van_Q;//存储货车的序号
    SqQueue ship_Q;//存储当前船上的情况

    InitQueue(van_Q);
    InitQueue(ship_Q);

    for(int i=1;i<=n;i++)
    {
        if(ship_tot<=9)//船有足够的空间(小于10)
        {
            if(a[i]==1)//客车
            {
                if(bus_tot>=4&&QueueEmpty(van_Q)==false)//货车队列有且当前客车数量不小于3
                {
                    int num;
                    DeQueue(van_Q, num);//出货车队
                    van_tot--;//货车队列中的数量减少
                    ship_tot++;//货车上船
                    EnQueue(ship_Q, num);

                    bus_tot=0;
                }

                if(ship_tot<=9)//由于有可能刚好把货车放入使得船满
                {
                    EnQueue(ship_Q, i);
                    bus_tot++;
```

```
                    ship_tot++;
                }
                else if(ship_tot==10)
                {
                    cout << "第" << ++ship_cnt << "辆船:";
                    PrintQueue(ship_Q);
                    ship_tot=0,bus_tot=0;
                    ClearQueue(ship_Q);

                    EnQueue(ship_Q, i);
                    bus_tot++;
                    ship_tot++;

                }
            }
            else//货车
            {
                EnQueue(van_Q, i);
                van_tot++;
            }
        }

        if(ship_tot==10)
        {
            cout << "第" << ++ship_cnt << "辆船:";
            PrintQueue(ship_Q);
            ship_tot=0,bus_tot=0;
            ClearQueue(ship_Q);
        }

        //cout << i << ": " << ship_tot << " " << bus_tot << " " << van_tot <<
endl;
    }


    while(QueueEmpty(van_Q)==false)
    {
        if(ship_tot<=9)
        {
            int num;
            DeQueue(van_Q, num);//出货车队
            van_tot--;//货车队列中的数量减少
            ship_tot++;//货车上船
            EnQueue(ship_Q, num);
        }
        else if(ship_tot==10)
        {
            cout << "第" << ++ship_cnt << "辆船:";
            PrintQueue(ship_Q);
            ship_tot=0;
            ClearQueue(ship_Q);

```

```
            int num;
            DeQueue(van_Q, num);//出货车队
            van_tot--;//货车队列中的数量减少
            ship_tot++;//货车上船
            EnQueue(ship_Q, num);
        }

        if(ship_tot==10)//此处注意细节
        {
            cout << "第" << ++ship_cnt << "辆船:";
            PrintQueue(ship_Q);
            ship_tot=0;
            ClearQueue(ship_Q);
        }
    }


    if(ship_tot<=9)
    {
        cout << "第" << ++ship_cnt << "辆船:";
        PrintQueue(ship_Q);
        ship_tot=0;
        ClearQueue(ship_Q);
    }
}
```