03. 假设以 I 和 O 分别表示入栈和出栈操作。栈的初态和终态均为空，入栈和出栈的操作序列可表示为仅由 I 和 O 组成的序列，可以操作的序列称为合法序列，否则称为非法序列。

1）下面所示的序列中哪些是合法的？ A,D

A. IOIIOIOO　　B. IOOIOIIO　　C. IIIOIOIO　　D. IIIOOIOO

2）通过对 1）的分析，写出一个算法，判定所给的操作序列是否合法。若合法，返回 true，否则返回 false（假定被判定的操作序列已存入一维数组中）。

```
bool judge(char a[]){
    int i=0;
    int j=0,k=0;
    while(a[i] != '\0'){
        switch(a[i]){
            case 'I': j++; break;
            case 'O': k++;
                if(k>j) { cout << "不合法"; return false;}.
        } // switch
        i++;
    }. // while
    if(j!=k) { cout << "不合法; return false;}.
    return true;
}.
```

04. 设单链表的表头指针为 L，结点结构由 data 和 next 两个域构成，其中 data 域为字符型。试设计算法判断该链表的全部 n 个字符是否中心对称。例如 xyx、xyyx 都是中心对称。

```
bool judge(LinkList L, int n){
    char stk[n/2];
    int i;
    LNode *p = L->next;
    for(i=0; i<n/2; i++){
        stk[i] = p->data;
        p = p->next;
    }
    i--;        // 计算后半部为剩余个数，若是奇数，不包括中心结点。
    if(n%2!=0)  p=p->next;
    while(p && stk[i]==p->data){
        p=p->next;
        i--;
    }
    if(i==-1)   return true;
    return false;
}
```

05. 设有两个栈 s1、s2 都采用顺序栈方式，并共享一个存储区[0,…,maxsize-1]，为了尽量利用空间，减少溢出的可能，可采用栈顶相向、迎面增长的存储方式。试设计 s1、s2 有关入栈和出栈的操作算法。


```
typedef struct stk{
        int  stack[10];
        int  top[2];
} stk;
stk s;

int push (int i, int num){    //i为栈号. num为入栈元素
    if(i<0 || i>1){
      cout << "栈号不对";
      return 0;
    }
    if(S.top[0]+1 == S.top[1]){
      cout << "栈满";
      return 0;
    }
    switch(i){
       case 0:  S.stack[++S.top[0]] = num;  return 1;
       case 1:  S.stack[--S.top[1]] = num;  return 1;
    }
}// end push.
```

接上.

```
int  pop(int i){
    if(i<0 || i>1){
        cout << "栈号有误";
        return -1;
    }.
    switch(i){
    case 0:
        if(s.top[0] == -1){
            cout << "栈空";
            return -1;
        }.
        return  s.stack[stop[0]--];
    case 1:
        if(s.top[1] == maxSize){
            cout << "栈空";
            return -1;
        }.
        return  s.stack[s.top[1]++];
    } // switch.
} // end pop.
```

01. 若希望循环队列中的元素都能得到利用，则需设置一个标志域 tag，并以 tag 的值为 0 或 1 来区分队头指针 front 和队尾指针 rear 相同时的队列状态是“空”还是“满”。试编写与此结构相应的入队和出队算法。


```
int push (SqQueue &Q, int x){
    if (Q.front == Q.rear && Q.tag == 1){
        cout << "队满";
        return -1;
    }
    Q.data[Q.rear] = x;
    Q.rear = (Q.rear+1) % maxSize;
    Q.tag = 1;
    return 1;
}

int pop (SqQueue& Q, int& num){
    if (Q.front == Q.rear && Q.tag == 0){
        cout << "队空";
        return -1;
    }
    num = Q.data[Q.front];
    Q.front = (Q.front+1) % maxSize;
    Q.tag = 0;
    return 1;
}
```

02. Q是一个队列，S是一个空栈，实现将队列中的元素逆置的算法。

```
Void reverse(Stack& S, Queue Q){
    while (Q.rear != Q.front)
        push(S, DeQueue(Q));
    while (S.top != -1){
        int x;
        pop(S,x);
        EnQueue(Q,x);
    }
}.
```

03. 利用两个栈 S1,S2 来模拟一个队列，已知栈的 4 个运算定义如下：

```
Push(S,x);                  //元素 x 入栈 S
Pop(S,x);                   //S 出栈并将出栈的值赋给 x
StackEmpty(S);              //判断栈是否为空
StackOverflow(S);           //判断栈是否满
```

如何利用栈的运算来实现该队列的 3 个运算（形参由读者根据要求自己设计）？

```
Enqueue;                    //将元素 x 入队
Dequeue;                    //出队，并将出队元素存储在 x 中
QueueEmpty;                 //判断队列是否为空
```

```
int Enqueue (Stack2 S1, Stack2 S2, int x){
    if (!StackOverflow(S1)){
        push(S1,x);
        return 1;
    }
    if(!StackEmpty(S2)){
        cout << "队满";
        return 0;
    }
    else {
        int a;
        while(!StackEmpty(S1)){
            pop(S1, a);
            push(S2,a);
        }
        Push(S1,x);
    } // else.
    return 1;
}
```

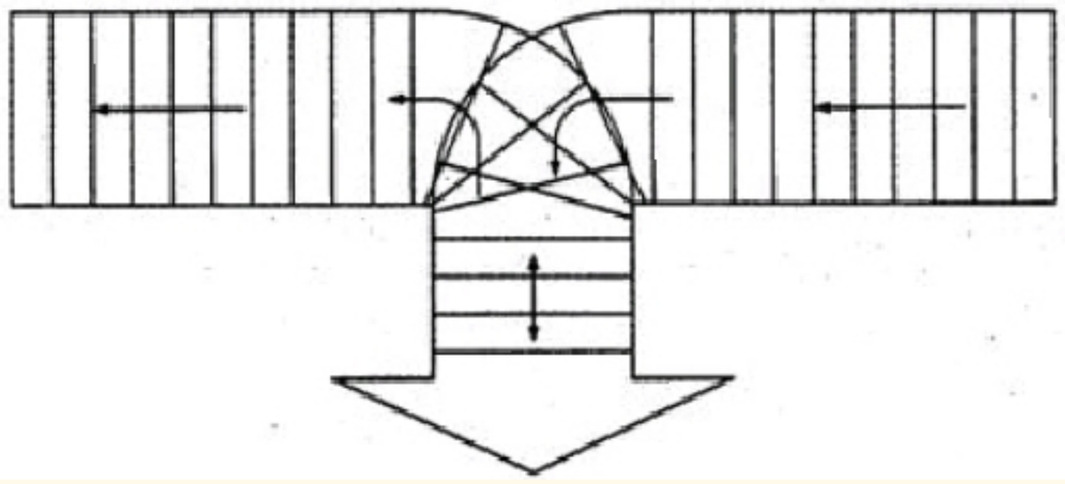
接上 void Dequeue(Stack &S1, Stack &S2, int& x){
if(!StackEmpty(S2))
pop(S2,x);
else if(StackEmpty(S1))
cout << "队空";
else {
int a;
while(!StackEmpty(S1)){
pop(S1, a);
push(S2, a);
}.
pop(S2, x);
}. //else.
}.

bool QueueEmpty(Stack S1, Stack S2){
return StackEmpty(S1)&& StackEmpty(S2);
}.

01. 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。

```
bool judge (char a[]){
    int i = 0;
    InitStack(S);
    while (a[i] != '\0'){
        switch (a[i]){
            case '(':
            case '[':
            case '{':
                Push(S,a[i]);
                i++;
                break;
            case ')': Pop(S,e);
                if(e != '(') return false;
            case ']': Pop(S,e);
                if(e != '[') return false;
            case '}': Pop(S,e);
                if(e != '{') return false;
            default : break;
        }. // switch.
    } // while.
    if (!StackEmpty(S))  return false;
    return true;
}.
```

02. 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。



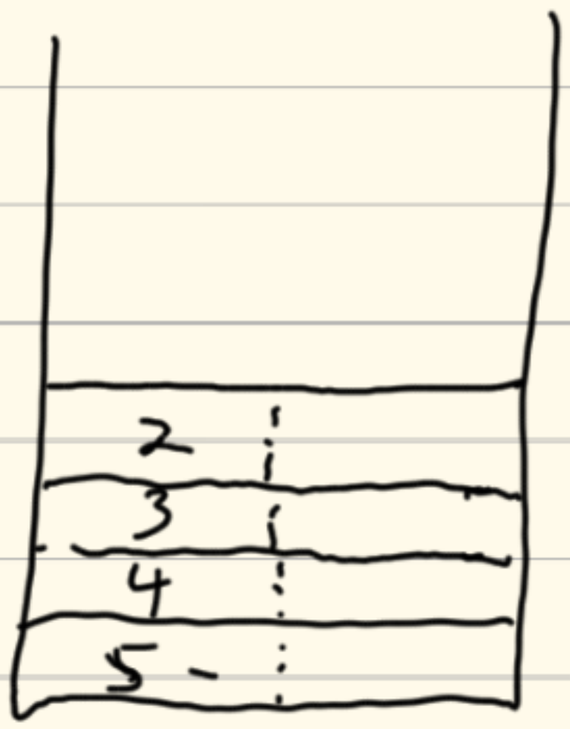
```
Void  manage (char train[]){                    // H入栈, S放前面.
    int i=0, P=0;                               // 出栈. 将H放入S后面.
    char c;
    stack S;
    InitStack(S);
    while (train[i] != '\0'){
        if(train[i] == 'H'){
            Push(S,'H');
        else  train[P++] = 'S';
        i++;
    }.
    while (! StackEmpty(S)){
        Pop(S,c);
        train[P++] = C;
    }.
}
```

03. 利用一个栈实现以下递归函数的非递归计算：

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

```
Struct stk{
    int no;        // 记录 n.
    double val;    // 记录计算结果
} s[maxSize];

double calc(int n, double x){
    double a=1. b=2*x; //记录前2个值.
    int top=-1, i;
    for(i=n; i>=2; i--)
        s[++top].no=i;        // 5, 4, 3, 2.
    while(top!=-1){
        s[top].val=2*x*b-2*(s[top].no-1)*a;
        a=b;
        b=s[top].val;
        top--;
    }.
    if(n==0) return a;        // a=0.
    return b;
}
```



04. 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。

```
Queue q, q1, q2;      //船,客车,货车.
void manage(){
    int i=0, j=0;     //上客车数,总车数.
    while(j<10){
        if(!QueueEmpty(q1) && i<4){
            DeQueue(q1, x);
            EnQueue(q, x);
            i++; j++;
        }
        else if(i==4 && !QueueEmpty(q2)){
            DeQueue(q2, x);
            EnQueue(q, x);
            j++; i=0;
        }
        else{
            while(j<10 && i<4 && !QueueEmpty(q2)){
                DeQueue(q2, x);
                EnQueue(q, x);
                i++; j++;
            }
            i=0;
        }
        if(QueueEmpty(q1) && QueueEmpty(q2))
            j=11;
    } // while.
}
```