3.1.1

> 有 5个元素，其入栈次序为 A, B, C, D, E，在各种可能的出栈次序中，第一个出栈元素为 C且第二个出栈元素为 D的出栈序列有哪几个?

- $CDEBA$
- $CDBEA$
- $CDBAE$

共有以上三种情况,由于 C第一个出栈,说明栈内有 AB(B为当前栈顶),又因为 D第二个出栈,说明 E还没入栈.

3.1.2

> 若元素的进栈序列为 A, B, C, D, E，运用栈操作，能否得到出栈序列 B, C, A, E, D和 D, B, A, C, E?为什么?

- B, C, A, E, D**可**得到， A, B进栈, B出栈, C进栈、出栈， D, E进栈, E出栈, D出栈.
- D, B, A, C, E**不可**得到,由于 D第一个出栈,说明 A, B, C已入栈,而第二出栈的元素是 B,而栈顶元素是 C且 B已入栈,故不可能得到该栈序列.

3.1.3

> 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. $IOIIOIOO$
>
> 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).

A, D合法, B, C不合法(B中先入栈一次,再出栈两次，即在空栈的时候出栈; D中最终的栈不为空,即出栈和入栈的次数不相同).

```
bool check_legality(string s)
{
    int cnt_o=0,cnt_i=0;

    for(int i=0;i<s.size();i++)
    {
        if(s[i]=='O')
            cnt_o++;
        else cnt_i++;
```

```
        if(cnt_o>cnt_i)
        {
            cout << "序列不合法" << endl;
            return false;
        }
    }
    if(cnt_o!=cnt_i)
    {
        cout << "序列不合法" << endl;
        return false;
    }
    else
    {
        cout << "序列合法" << endl;
        return true;
    }
}
```

3.1.4

> 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。

利用栈的先进后出的特性,设置两个快慢指针 q和 p,用来找到中间结点,将 p之后的结点一次入栈, q指针重新指向第一个结点,然后与栈内元素一一比较,若存在不同,则不对称.(此处由于考虑快慢指针的原因不用讨论奇偶关系)

```
bool check_symmetry(LinkList L)
{
    LiStack S;
    InitStack(S);

    LNode *p=L,*q=L;

    while(q->next!=NULL)
    {
        p=p->next;
        q=q->next;
        if(q->next!=NULL)
            q=q->next;
    }

    q=p->next;//让中间节点的后半部分入栈

    while(q!=NULL)
    {
        Push(S, q->data);
        q=q->next;
    }


    q=L->next;
```

```
    while(StackEmpty(S)==0)//栈不为空
    {
        char c;
        GetTop(S, c);
        if(q->data!=c)
            break;

        q=p->next;
        Pop(S, c);
    }

    if(StackEmpty(S))
    {
        cout << "中心对称" << endl;
        DestroyStack(S);
        return true;
    }
    else
    {
        cout << "非中心对称" << endl;
        DestroyStack(S);
        return false;
    }
}
```

3.1.5

> 设有两个栈 s_1、 s_2都采用顺序栈方式，并共享一个存储区 $[0,\cdots,maxsize-1]$，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 s_1、 s_2有关入栈和出栈的操作算法。

```
bool Push(SqStack &S,ElemType x,int type)//进栈
{
    if(type!=1&&type!=2)//栈号不对
        return false;
    if(S.top1+1==S.top2)//栈满
        return false;
    if(type==1)
        S.data[++S.top1]=x;
    else
        S.data[--S.top2]=x;
    return true;
}


bool Pop(SqStack &S,ElemType &x,int type)//出栈
{
    if(type!=1&&type!=2)//栈号不对
        return false;
    if(StackEmpty(S, type)==true)//栈空
```

```
        return false;

    if(type==1)
        x=S.data[S.top1--];
    else
        x=S.data[S.top2++];
    return true;
}

bool GetTop(SqStack S,ElemType &x,int type)//获取栈顶
{
    if(type!=1&&type!=2)//栈号不对
        return false;
    if(StackEmpty(S, type)==true)//栈空
        return false;
    if(type==1)
        x=S.data[S.top1];
    else x=S.data[S.top2];
    return true;
}
```