1.//P45 第 8 题：已知 A、B 和 C 为三个元素值递增有序排列的顺序表，现要求对表 A 作如下运算：删除那些既在 B 中出现又在 C 中出现的元素。

```
#include<stdio.h>
#include<malloc.h>
typedef int datatype;
#define maxsize 1024
typedef struct
{
    datatype data[maxsize];
    int last;
} sequenlist;
sequenlist* create();
void print(sequenlist*);
void Delete(sequenlist*, sequenlist*, sequenlist*);
int main(void)
{
    sequenlist*LA, *LB, *LC;
    printf("*************please input LA numbers : **********************\n");
    LA = create();
    printf("*************please input LB numbers : **********************\n");
    LB = create();
    printf("*************please input LC numbers : **********************\n");
    LC = create();
    printf("*************删除之前的 LA 表为: **********************\n");
    print(LA);
    Delete(LA, LB, LC);
    printf("*************删除之后的 LA 表为: **********************\n");
    print(LA);
    return 0;
}

//建立顺序表
sequenlist* create()
{
    sequenlist* L;
    L = (sequenlist*)malloc(sizeof(sequenlist));
    L->last = 0;
    int ch;
    scanf("%d", &ch);
    while (ch != -1)
    {
        L->last++; //从 1 单元开始存放元素
        L->data[L->last] = ch;
```

```
            scanf("%d", &ch);
        }
        return L;
}

void Delete(sequenlist *LA, sequenlist *LB, sequenlist *LC)
{



}/*SqList_Delete*/


//输出顺序表
void print(sequenlist*L)
{
        for (int i = 1; i <= L->last; i++)
            printf("%d\n", L->data[i]);
}
```

2.//P45 第 8 题：已知 A、B 和 C 为三个元素值递增有序排列的链表，现要求对表 A 作如下运算：删除那些既在 B 中出现又在 C 中出现的元素。

```
#include <stdio.h>
#include<malloc.h>
typedef struct node
{
   int data;
   struct node *next;
}LinkList;/*定义链表节点的结构*/

LinkList* create();/*函数声明*/
void ListDelete(LinkList * La, LinkList * Lb, LinkList *Lc);
void Print(LinkList *);

int main (void)
{/*功能：建立单链表 A,B,C，并且删除 A 中均在 B 和 C 中出现的数据。*/
      LinkList *headA, *headB, *headC;
      headA = create();/*建立链表*/
      headB = create();
      headC = create();
     printf("**********删除之前的 3 个表分别为: **********\n");
      Print(headA);/*输出显示链表数据*/
      Print(headB);
      Print(headC);
      ListDelete(headA, headB, headC);/*删除 A 中满足条件的节点*/
      printf("**********删除之后的 LA 表为: **********\n");
      Print(headA);
      return 0;
}

void ListDelete(LinkList *La, LinkList *Lb, LinkList *Lc)
{/*功能：删除 A 中的有关结点*/




}
```

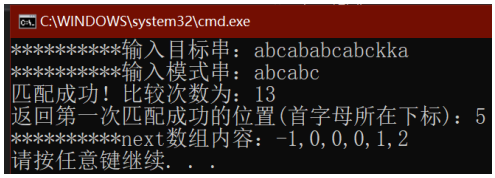
3.//P74 第 4 题：利用栈的基本运算将栈 S 中值为 m 的元素全部删除。

```
#include<stdio.h>
#include<malloc.h>
#define maxsize 1024
typedef char datatype;
typedef    struct stack
{
      datatype data[maxsize];
      int Top;
} seqstack;

seqstack * Initstack();
void Delete(seqstack *, datatype);
void Push(seqstack *, datatype);
datatype Pop(seqstack *);
int EmptyS(seqstack *s);
void Print(seqstack * s);
int main (void)
{
      seqstack *s;
      datatype m;
      printf("**********请输入 m: **********\n");
      scanf("%c", &m);
      printf("**********请初始化栈 s: **********\n");
      s = Initstack();
      printf("**********输出栈 s: **********\n");
      Print(s);
      Delete(s, m);//调用删除函数
      printf("\n**********  输 出 删 除 m 之 后 的 栈 s:
**********\n");
      Print(s);
      return 0;

}

void Delete(seqstack *s,datatype m)
{




}
```

```
seqstack* Initstack ( )
{




}
void Push(seqstack *s, datatype e)
{




}

int EmptyS(seqstack *s)
{




}


datatype Pop(seqstack *s)
{




}

void Print(seqstack*s)
{


}
```

4. //改进的模式匹配追踪算法（KMP）测试

以类似下图的形式展示结果：



测试用例 1：

输入:
目标串: abcababcabc
模式串: abcabc

输出:
匹配成功！比较次数为：13
返回第一次匹配成功的位置(首字母位序)：5
*********next 数组内容：-1,0,0,0,1,2

测试用例 2：

输入:
目标串: abcababcabckka
模式串: abcabc

输出:
匹配成功！比较次数为：13
返回第一次匹配成功的位置(首字母位序)：5
*********next 数组内容：-1,0,0,0,1,2

测试用例 3：

输入:
目标串: abcababcabckka
模式串: abcabd

输出:
匹配失败！ 比较次数为：20
*********next 数组内容：-1,0,0,0,1,2

测试用例 4：

输入:
目标串: aaaaaaaaaaaaaaaaaakuu
模式串: aaaaak

输出:
匹配成功！比较次数为：34
返回第一次匹配成功的位置(首字母位序)：14
*********next 数组内容：-1,0,1,2,3,4

测试用例 5：

输入:
目标串: aaaaaaaaaaaaaaaaaak
模式串: aaaaau

输出:
匹配失败！ 比较次数为：40
*********next 数组内容：-1,0,1,2,3,4

5. P94 第 6 题 ：利用 C 的库函数 strlen、strcpy 和 strncpy 写一算法 void StrDelete(char*S,int i,int m)，删去串 S 中从位置 i 开始的连续 m 个字符。若 i≥strlen(S)，则没有字符被删除；若 i+m≥strlen(S)，则将 S 中从位置 i 开始直至末尾的字符均删去。

```
#include "stdafx.h"
#include<stdio.h>
#include<string.h>
#include<malloc.h>
//顺序串的结构类型定义
#define maxsize 100
typedef struct
{
    char str[maxsize];
    int len;
}seqstring;

void strPut(seqstring*);
void strDelete(seqstring*,int,int);
int main(void)
{
    seqstring*S;
    int i,m;
    S=(seqstring*)malloc(sizeof(seqstring));
    printf("**********请输入字符串**********：");
    gets(S->str);
    S->len=strlen(S->str);
    printf("**********输入的字符串显示如下**********");
    strPut(S);
    printf("**********输入字符串的长度为：%d \n", S->len);
    printf("********** 请 输 入 删 除 字 符 的 开 始 位 置 ：**********\n");
    scanf("%d",&i);
    printf("\n********** 请 输 入 要 删 除 的 字 符 个 数 **********\n");
    scanf("%d",&m);
    strDelete(S,i,m);
    printf("**********删除子串后的字符串为**********\n");
    strPut(S);
    return 0;
}
```

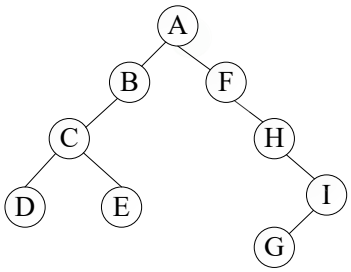
测试用例参考：（13 个字符为例）
输入字符串：abdeuyqwxzkjk
删除的开始位置：9
删除的字符个数：2
删除子串后的字符串: abdeuyqwkjk

测试用例参考：（13 个字符为例）
输入字符串：abdeuyqwxzkjk
删除的开始位置：9
删除的字符个数：8
删除子串后的字符串: abdeuyqw

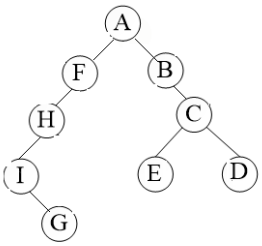
6. //交换左右子树的递归算法实现（二叉树翻转）

题目要求：已知二叉树采用二叉链表存储结构，编写一个算法交换二叉树所有左、右子树的位置，即结点的左子树变为结点的右子树，右子树变为左子树。

```
#include "stdafx.h"
#include<stdio.h>
#include<malloc.h>
//二叉链表的结构类型定义
const int maxsize=1024;
typedef char datatype;
typedef struct node
{
    datatype data;
    struct node *lchild,*rchild;
}bitree;

bitree*creattree();
void preorder(bitree*);
void inorder(bitree *);
void swap(bitree*);

int main(void)
{
    bitree*pb;
    printf("**********按层次输入二叉树，虚结点输入'@'，以'#'结束输入********\n");
    pb=creattree();
    printf("******************交换之前的原二叉树先序遍历序列为：");
    preorder(pb);
    printf("\n");
    printf("******************交换之前的原二叉树中序遍历序列为：");
    inorder(pb);
    printf("\n");
    pb=swap(pb);
    printf("******************交换之后的二叉树先序遍历序列为：");
    preorder(pb);
    printf("\n");
    printf("******************交换之后的二叉树中序遍历序列为：");
    inorder(pb);

    printf("\n");
    return 0;
}
```



假设构造的二叉树如图所示：



交换左右子树后的二叉树如图所示：

输出如下结果：

交换之前的原二叉树先序遍历序列为：A B C D E F H I G

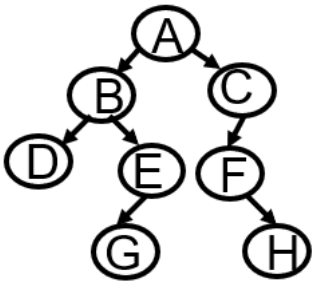
交换之前的原二叉树中序遍历序列为：D C E B A F H G I

交换之后的二叉树先序遍历序列为：A F H I G B C E D

交换之后的二叉树先序遍历序列为：I G H F A B E C D

```
**********按层次输入二叉树，虚结点输入'@'，以'#'结束输入********
ABFC@@HDE@@@@@I@@@@@@@@@@@@@@G#
********************交换之前的原二叉树先序遍历序列为：A  B  C  D  E  F  H  I  G
********************交换之前的原二叉树中序遍历序列为：D  C  E  B  A  F  H  G  I
********************交换之后的二叉树先序遍历序列为：A  F  H  I  G  B  C  E  D
********************交换之后的二叉树中序遍历序列为：I  G  H  F  A  B  E  C  D
```

7. 以中序遍历为基础,写出在二叉树上查找指定结点 x 的中序前驱结点的算法。



例如二叉树如图所示：

测试用例 1：
输入：待查找结点为 A
输出：结点 A 的中序前驱结点为 E

测试用例 2：
输入：待查找结点为 C
输出：结点 C 的中序前驱结点为 H

测试用例 3：
输入：待查找结点为 D
输出：结点 D 没有中序前驱结点

测试用例 4：
输入：待查找结点为 E
输出：结点 E 的中序前驱结点为 G

测试用例 5：
输入：待查找结点为 G
输出：结点 G 的中序前驱结点为 B

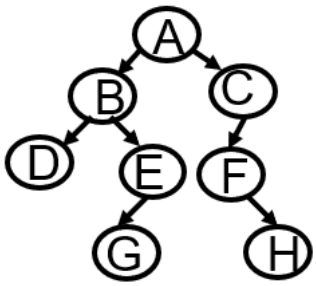
其它可选测试用例：

输入：待查找结点为 B
输出：结点 B 的中序前驱结点为 D

输入：待查找结点为 F
输出：结点 F 的中序前驱结点为 A

输入：待查找结点为 H
输出：结点 H 的中序前驱结点为 F

以中序遍历为基础，写出在二叉树上查找指定结点 x 的中序后继结点的算法。



例如二叉树如图所示：

测试用例 1：
输入：待查找结点为 A
输出：结点 A 的中序后继结点为 F

测试用例 2：
输入：待查找结点为 B
输出：结点 B 的中序后继结点为 G

测试用例 3：
输入：待查找结点为 G
输出：结点 G 的中序后继结点为 E

测试用例 4：
输入：待查找结点为 E
输出：结点 E 的中序后继结点为 A

测试用例 5：
输入：待查找结点为 C
输出：结点 C 没有中序后继结点

其它可选测试用例：

输入：待查找结点为 D
输出：结点 D 的中序后继结点为 B

输入：待查找结点为 F
输出：结点 F 的中序后继结点为 H

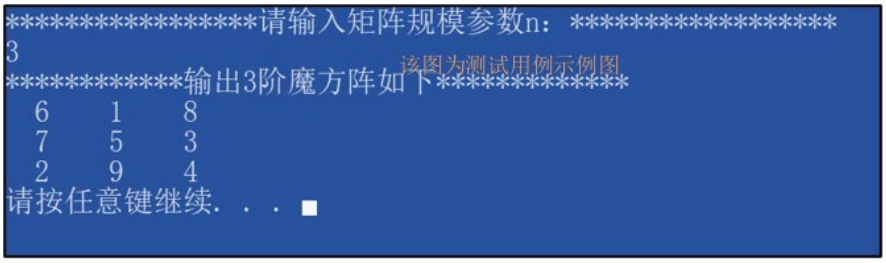
输入：待查找结点为 H
输出：结点 H 的中序后继结点为 C

9. n 阶魔方阵（n 为奇数）的构造算法测试

要求可以从键盘灵活输入 n=3,5,7,9……

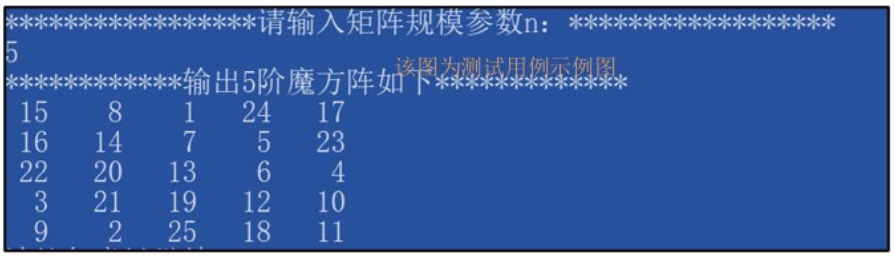
输入 n=3 时，输出魔方阵：

6	**1**	**8**
7	**5**	**3**
2	**9**	**4**

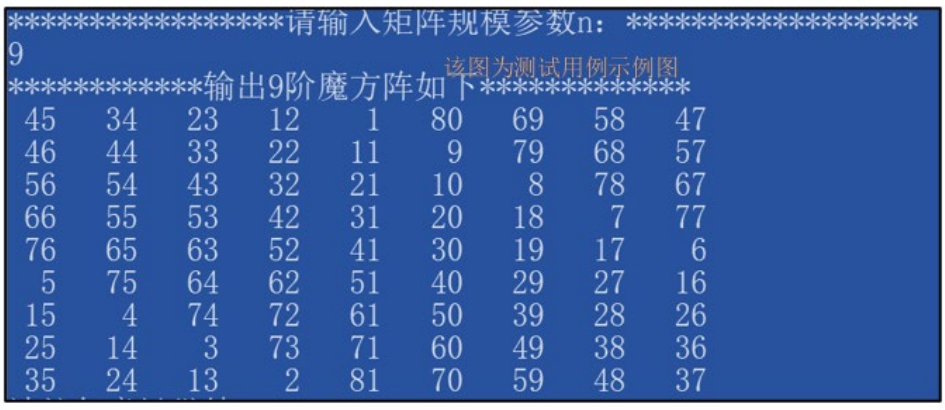


输入 n=5 时，输出魔方阵：

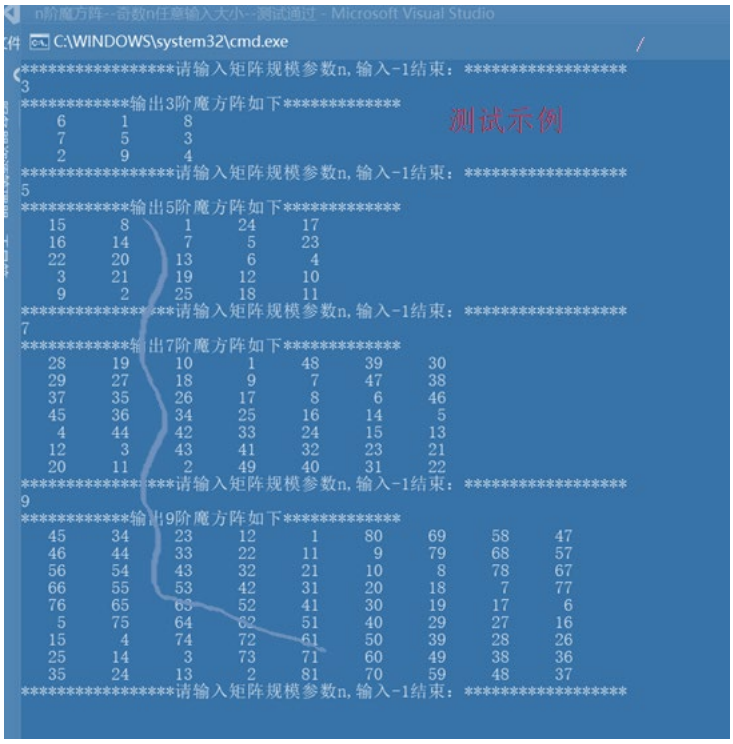
15	**8**	**1**	**24**	**17**
16	**14**	**7**	**5**	**23**
22	**20**	**13**	**6**	**4**
2	**21**	**19**	**12**	**10**
9	**2**	**25**	**18**	**11**



输入 n=9 时，输出如下：



或按如下形式输出


n阶魔方阵--奇数n任意输入大小--测试通过 - Microsoft Visual Studio
C:\WINDOWS\system32\cmd.exe
**********请输入矩阵规模参数n,输入-1结束：**********
3
**********输出3阶魔方阵如下**********
6 1 8
7 5 3
2 9 4
**********请输入矩阵规模参数n,输入-1结束：**********
5
**********输出5阶魔方阵如下**********
15 8 1 24 17
16 14 7 5 23
22 20 13 6 4
3 21 19 12 10
9 2 25 18 11
**********请输入矩阵规模参数n,输入-1结束：**********
7
**********输出7阶魔方阵如下**********
28 19 10 1 48 39 30
29 27 18 9 7 47 38
37 35 26 17 8 6 46
45 36 34 25 16 14 5
4 44 42 33 24 15 13
12 3 43 41 32 23 21
20 11 2 49 40 31 22
**********请输入矩阵规模参数n,输入-1结束：**********
9
**********输出9阶魔方阵如下**********
45 34 23 12 1 80 69 58 47
46 44 33 22 11 9 79 68 57
56 54 43 32 21 10 8 78 67
66 55 53 42 31 20 18 7 77
76 65 63 52 41 30 19 17 6
5 75 64 62 51 40 29 27 16
15 4 74 72 61 50 39 28 26
25 14 3 73 71 60 49 38 36
35 24 13 2 81 70 59 48 37
**********请输入矩阵规模参数n,输入-1结束：**********
测试示例