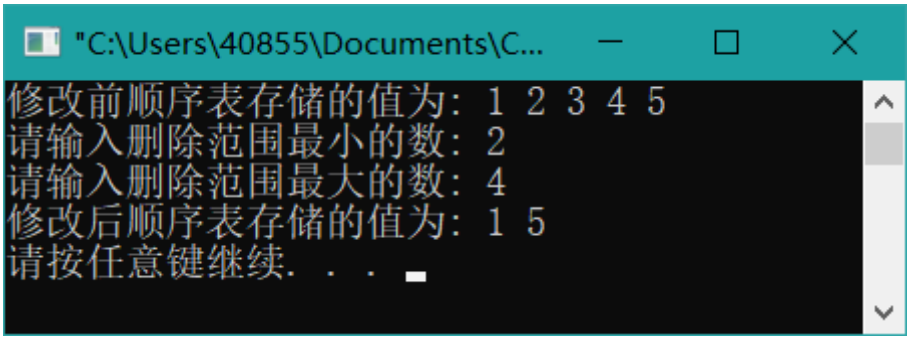
第二章

6、

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 30

typedef int ElemType;
typedef struct
{
    ElemType data[MAXSIZE];
    int length;
} SqList;

void fun(SqList *&L, int x, int y)
{
    int i, j=0;
    for (i = 0; i < L->length; i++)
    {
        if (L->data[i] < x || L->data[i] > y)
        {
            L->data[j] = L->data[i];
            j++;
        }
    }
    L->length=j;
}

void InitList(SqList *&L)
{
    int i;
    int a[5] = {1, 2, 3, 4, 5};

    L = (SqList *)malloc(sizeof(SqList));

    for (i = 0; i < 5; i++)
        L->data[i] = a[i];
    L->length = 5;
}

void DispList(SqList *L)
{
    int i;

    for (i = 0; i < L->length; i++)
```

```
        printf("%d ", L->data[i]);
    printf("\n");

}

int main (void)
{
    SqList *L;
    int low, high;

    InitList (L);
    printf("修改前顺序表存储的值为: ");
    DispList (L);
    printf("请输入删除范围最小的数: ");
    scanf("%d", &low);
    printf("请输入删除范围最大的数: ");
    scanf("%d", &high);

    fun (L, low, high);
    printf("修改后顺序表存储的值为: ");
    DispList (L);

    return 0;
}
```

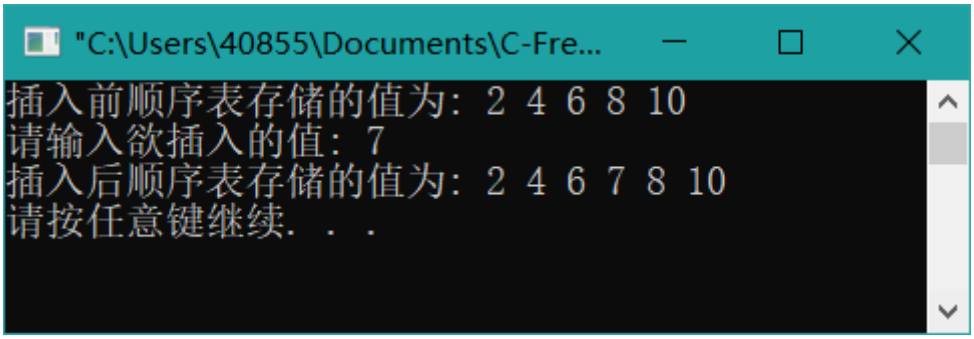


7、

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 30

typedef int ElemType;
typedef struct
{
    ElemType data[MAXSIZE];
    int length;
```

```
}SqList;

void Insert (SqList *L, ElemType x)
{
	int i=0, j=0;

	while (L->data[i] <= x && i < L->length)
		i++;
	for (j = L->length; j > i; j--)
		L->data[j] = L->data[j-1];

	L->data[i] = x;
	L->length++;
}

void InitList (SqList *&L)
{
	int i;
	int a[5] = {2, 4, 6, 8, 10};

	L = (SqList *)malloc(sizeof(SqList));
	for (i = 0; i < 5; i++)
		L->data[i] = a[i];
	L->length = 5;
}

void DispList (SqList *L)
{
	int i;

	for (i = 0; i < L->length; i++)
		printf ("%d ", L->data[i]);
	printf ("\n");
}

int main (void)
{
	SqList *L;
	int n;

	InitList (L);
	printf("插入前顺序表存储的值为: ");
	DispList (L);
```

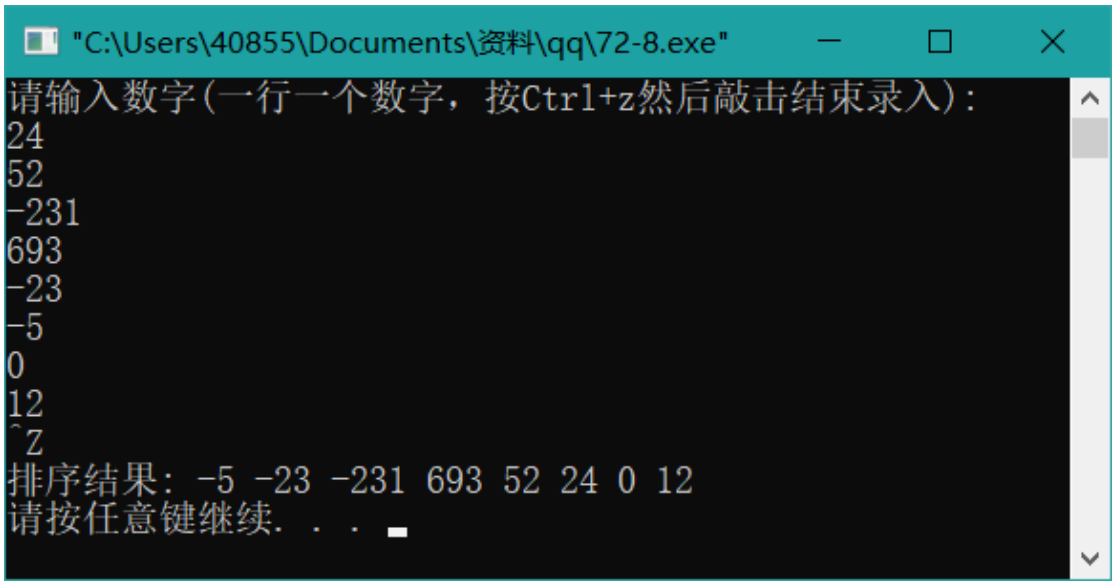
```
    printf("请输入欲插入的值: ");
    scanf("%d", &n);

    Insert (L, n);
    printf("插入后顺序表存储的值为: ");
    DispList (L);

    return 0;
}
```



8、

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 20

typedef struct {
    int data[MAXSIZE];
    int length;
} Sqlist;

void CreateList(Sqlist*& L, int a[], int n)
{
    int i = 0;
    L = (Sqlist*)malloc(sizeof(Sqlist));

    while(i < n)
    {
        L->data[i] = a[i];
        i++;
    }
    L->length = n;
}

void Displist(Sqlist* L)
{
```

```
    int i;
    for(i=0; i < L->length; i++)
        printf("%d ", L->data[i]);
    printf("\n");
}

void arrange(Sqlist*& L)
{
    int i = 0, j = L->length-1, n;
    while (i<j)
    {
        while (L->data[i]<0)
            i++;
        while (L->data[j]>=0)
            j--;

        if (i<j) {
            n = L->data[j];
            L->data[j] = L->data[i];
            L->data[i] = n;
        }
    }
}

int main(void)
{
    int a[MAXSIZE], i = 0, x;
     Sqlist* list;

    printf("请输入数字(一行一个数字，按 Ctrl+z 然后敲击结束录入):\n");

    while(scanf("%d", &a[i]) != EOF){
        i++;
     }

    CreateList(list, a, i);

    arrange(list);

    printf("排序结果: ");
    Displist(list);

    return 0;
}
```



11、

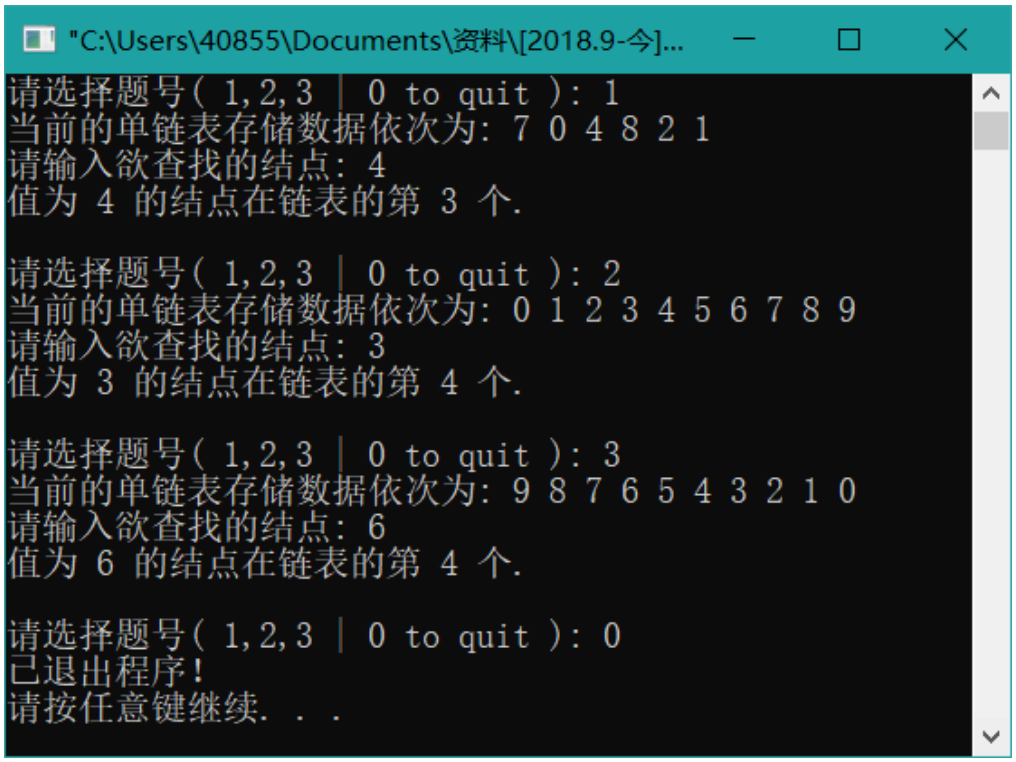
```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 6

typedef int ElemType;
typedef struct LNode
{
    ElemType data;
    struct LNode *next;
} LinkNode;

bool LocateElem_1(LinkNode*& L, ElemType x, ElemType &n)
{
    int i = 1;
    LinkNode *p = L->next;
    while (p->data != x && p != NULL)
    {
        p = p->next;
        i++;
    }
    if (p == NULL)
        return false;
    else
        n = i;
        return true;
}
```

```
bool LocateElem_2(LinkNode*& L, ElemType x, ElemType &n)
{
    int i = 1;
    LinkNode *p = L->next;
    while (p->data < x && p != NULL)
    {
        p = p->next;
        i++;
    }
    if (p == NULL || p->data > x)
        return false;
    else
        n = i;
        return true;
}

bool LocateElem_3(LinkNode*& L, ElemType x, ElemType &n)
{
    int i = 1;
    LinkNode *p = L->next;
    while (p->data > x && p != NULL)
    {
        p = p->next;
        i++;
    }
    if (p == NULL || p->data < x)
        return false;
    else
        n = i;
        return true;
}

void InitList_1(LinkNode *&L)
{
    int i;
    int a[MAXSIZE] = {1, 2, 8, 4, 0, 7};
    LinkNode *p;

    L = (LinkNode *)malloc(sizeof(LinkNode));
    L->next = NULL;

    for (i = 0; i <MAXSIZE; i++)
    {
        p = (LinkNode *)malloc(sizeof(LinkNode));
```

```
        p->data = a[i];
        p->next = L->next;
        L->next = p;
    }
}

void InitList_2(LinkNode *&L)
{
    int i;
    int a[10] = {9, 8, 7, 6, 5, 4, 3, 2, 1};
    LinkNode *p;

    L = (LinkNode *)malloc (sizeof (LinkNode));
    L->next = NULL;

    for (i = 0; i <10; i++)
    {
        p = (LinkNode *)malloc (sizeof (LinkNode));
        p->data = a[i];
        p->next = L->next;
        L->next = p;
    }
}

void InitList_3(LinkNode *&L)
{
    int i;
    int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    LinkNode *p;

    L = (LinkNode *)malloc (sizeof (LinkNode));
    L->next = NULL;

    for (i = 0; i <10; i++)
    {
        p = (LinkNode *)malloc (sizeof (LinkNode));
        p->data = a[i];
        p->next = L->next;
        L->next = p;
    }
}

void DispList(LinkNode *L)
{
```

```
    L = L->next;
    while(L != NULL)
    {
        printf("%d ", L->data);
        L = L->next;
    }
    printf("\n");
}

int main (void)
{
    LinkNode *L;
    int n, num, choice;

    loop:
    printf("请选择题号( 1,2,3 | 0 to quit ): ");
    scanf("%d", &choice);
    switch (choice){
        case 1: goto loop1;
        case 2: goto loop2;
        case 3: goto loop3;
        case 0: printf("已退出程序！ \n"); return 0;
        printf("不存在该题！ ");
        return 0;
    }

    //Q1
    loop1:
    InitList_1(L);
    printf("当前的单链表存储数据依次为: ");
    DispList(L);
    printf("请输入欲查找的结点: ");
    scanf("%d", &n);
    if (LocateElem_1(L, n, num))
        printf("值为 %d 的结点在链表的第 %d 个.\n\n", n, num);
    else
        printf("不存在该数据结点!\n\n");

    goto loop;

    //Q2
    loop2:
    InitList_2(L);
    printf("当前的单链表存储数据依次为: ");
```

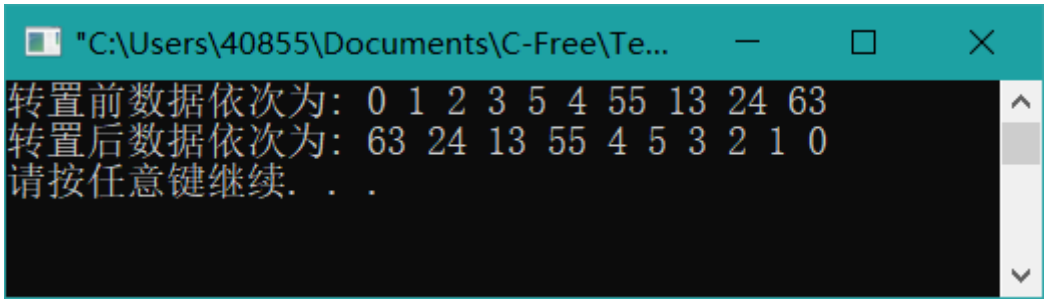
```
    DispList(L);
    printf("请输入欲查找的结点: ");
    scanf("%d", &n);
    if (LocateElem_2(L, n, num))
        printf("值为 %d 的结点在链表的第 %d 个.\n\n", n, num);
    else
        printf("不存在该数据结点!\n\n");

    goto loop;

    //Q3
    loop3:
    InitList_3(L);
    printf("当前的单链表存储数据依次为: ");
    DispList(L);
    printf("请输入欲查找的结点: ");
    scanf("%d", &n);
    if (LocateElem_3(L, n, num))
        printf("值为 %d 的结点在链表的第 %d 个.\n\n", n, num);
    else
        printf("不存在该数据结点!\n\n");

    goto loop;
}
```



12、

```
#include <stdio.h>
#include <stdlib.h>

typedef int ElemType;

typedef struct LNode
{
	ElemType data;
	struct LNode *next;
} LinkNode;

void reverse(LinkNode *& L)
{
	LinkNode *p = L->next, *q;
	L->next = NULL;
	while (p != NULL)
	{
		q = p->next;
		p->next = L->next;
		L->next = p;
		p = q;
	}
}


void InitList (LinkNode *&L)
{
	int i;
	int a[10] = {63, 24, 13, 55, 4, 5, 3, 2, 1, 0};
	LinkNode *p;

	L = (LinkNode *)malloc (sizeof (LinkNode));
	L->next = NULL;

	for (i = 0; i <10; i++)
	{
		p = (LinkNode *)malloc (sizeof (LinkNode));
		p->data = a[i];
		p->next = L->next;
		L->next = p;
	}
}
```

```
void DispList (LinkNode *L)
{
    LinkNode *p = L->next;

    while (p != NULL)
    {
        printf ("%d ", p->data);
        p = p->next;
    }
    printf ("\n");
}

int main (void)
{
    LinkNode *L;

    InitList (L);
    printf("转置前数据依次为: ");
    DispList (L);

    reverse (L);
    printf("转置后数据依次为: ");
    DispList (L);

    return 0;
}
```

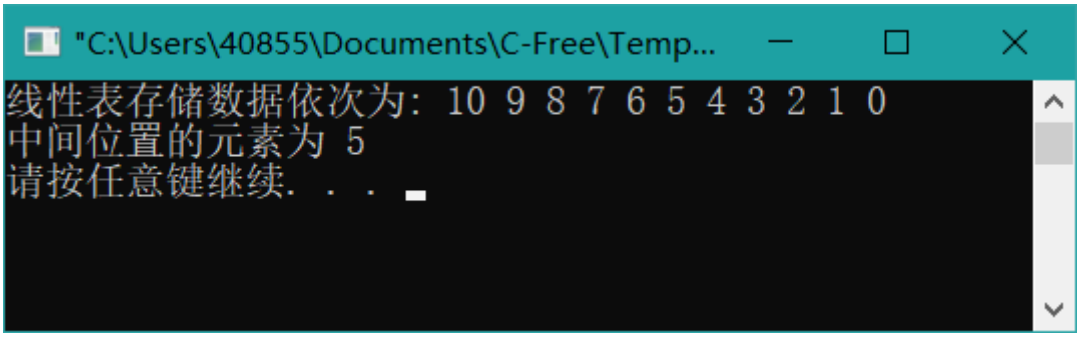


13、

```
#include <stdio.h>
#include <stdlib.h>

typedef int ElemType;

typedef struct LNode
```

```
{
    ElemType data;
    struct LNode *next;
} LinkNode;

ElemType search (LinkNode *L)
{
    LinkNode *p = L->next, *q = p;
    while (p->next != NULL && p->next->next != NULL)
    {
        p = p->next->next;
        q = q->next;
    }
    return q->data;
}

void InitList (LinkNode *&L)
{
    int a[11] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int i;

    L = (LinkNode *)malloc(sizeof (LinkNode));
    L->next = NULL;

    LinkNode *p;
    for   (i = 0; i < 11; i++)
    {
        p = (LinkNode *)malloc (sizeof(LinkNode));
        p->data = a[i];
        p->next = L->next;
        L->next = p;
    }
}

void DispList(LinkNode *L)
{
    L = L->next;
    while(L != NULL)
    {
        printf("%d ", L->data);
        L = L->next;
    }
    printf("\n");
}
```

```
int main (void)
{
    LinkNode *L;

    InitList (L);
    printf("线性表存储数据依次为: ");
    DispList(L);
    printf ("中间位置的元素为  %d\n", search(L));

    return 0;
}
```

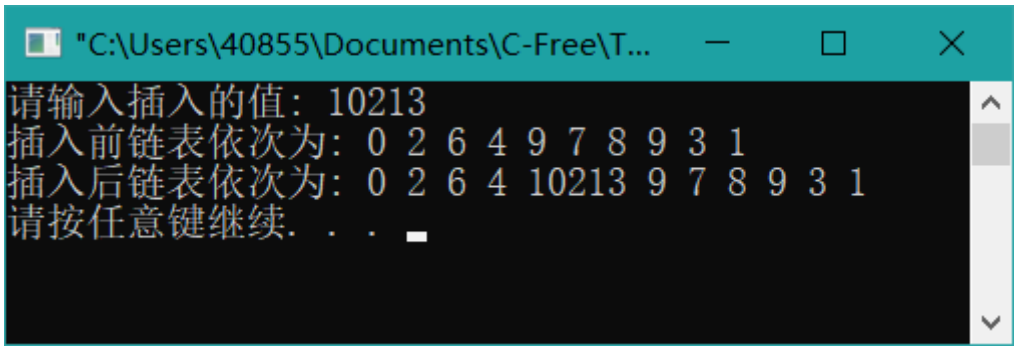


14、

```
#include <stdio.h>
#include <stdlib.h>

typedef int ElemType;

typedef struct LNode
{
    ElemType data;
    struct LNode *next;
}LinkNode;

void Insert (LinkNode *&L, ElemType x)
{
    LinkNode *p = L->next, *pre = L, *max = p, *maxpre = pre, *s;

    while (p != NULL)
    {
        if (max->data < p->data)
        {
            max = p;
            maxpre = pre;
```

```
            }
            pre = pre->next;
            p = p->next;
        }

        s = (LinkNode *)malloc(sizeof(LinkNode));
        s->data = x;
        s->next = max;
        maxpre->next = s;
}

void InitList (LinkNode *&L)
{
        int a[10] = {1, 3, 9, 8, 7, 9, 4, 6, 2, 0};
        int i;

        L = (LinkNode *)malloc(sizeof (LinkNode));
        L->next = NULL;

        LinkNode *p;
        for    (i = 0; i < 10; i++)
        {
            p = (LinkNode *)malloc (sizeof(LinkNode));
            p->data = a[i];
            p->next = L->next;
            L->next = p;
        }
}

void DispList (LinkNode *L)
{
        LinkNode *p = L->next;

        while (p != NULL)
        {
            printf ("%d ", p->data);
            p = p->next;
        }
        printf ("\n");
}


int main (void)
{
```

```
    ElemType x;
    LinkNode *L;

    printf("请输入插入的值: ");
    scanf("%d", &x);

    InitList (L);
    printf("插入前链表依次为: ");
    DispList (L);

    Insert (L, x);
    printf("插入后链表依次为: ");
    DispList (L);

    return 0;
}
```



15、

```
#include <stdio.h>
#include <stdlib.h>

typedef int ElemType;

typedef struct LNode
{
    ElemType data;
    struct LNode *next;
}LinkNode;

void sorted(LinkNode *&L)
{
    LinkNode *p = L->next->next, *pre, *q;
    L->next->next = NULL;
```

```
    while (p != NULL)
    {
        q = p->next;
        pre = L;
        while (pre->next != NULL && pre->next->data > p->data)
            pre = pre->next;
        p->next = pre->next;
        pre->next = p;
        p = q;
    }
}

void DestoryList (LinkNode *&L)
{
    LinkNode *pre = L, *p = L->next;

    while (p != NULL)
    {
        free(pre);
        pre = p;
        p = p->next;
    }
    free(pre);

    printf ("链表已销毁!\n");
}

void InitList (LinkNode *&L)
{
    int a[10] = {4, 3, 7, 8, 9, 5, 1, 6, 2, 0};
    int i;

    L = (LinkNode *)malloc(sizeof (LinkNode));
    L->next = NULL;

    LinkNode *p;
    for   (i = 0; i < 10; i++)
    {
        p = (LinkNode *)malloc (sizeof(LinkNode));
        p->data = a[i];
        p->next = L->next;
        L->next = p;
    }
}
```

```
void DispList (LinkNode *L)
{
    LinkNode *p = L->next;

    while (p != NULL)
    {
        printf ("%d ", p->data);
        p = p->next;
    }
    printf ("\n");
}

int main (void)
{
    LinkNode *L;

    InitList (L);
    printf("原链表存储值依次为: ");
    DispList (L);

    sorted(L);
    printf("递减排序后链表存储值依次为: ");
    DispList (L);

    DestoryList (L);

    return 0;
}
```

16、

```
#include<stdio.h>
#include<stdlib.h>
```

```
typedef int ElemType;

typedef struct Node
{
    ElemType data;
    int freq;
    struct Node *prior;
    struct Node *next;
} LinkNode;

void InitList(LinkNode **L, ElemType a[], int n)
{
    LinkNode *p, *q;
    (*L) = (LinkNode*)malloc(sizeof(LinkNode));
    (*L)->prior = NULL;
    (*L)->next = NULL;
    q = (*L);
    q->data = a[0];
    q->freq = 0;
    for (int i = 1; i < n; i++)
    {
        p = (LinkNode*)malloc(sizeof(LinkNode));
        p->data = a[i];
        p->freq = 0;
        p->next = NULL;
        q->next = p;
        p->prior = q;
        q = p;
    }
}

void LocateNode(LinkNode *L, ElemType x)
{
    if (L->next == NULL)
        return;

    LinkNode *p = L, *q;
    ElemType data;
    int freq;

    while (p!=NULL)
    {
        if (p->data == x)
        {
```

```
                p->freq++;
                q = p;
                while (q->prior != NULL&&q->prior->freq<q->freq)
                {
                    data = q->data;
                    freq = q->freq;
                    q->data = q->prior->data;
                    q->freq = q->prior->freq;
                    q->prior->data = data;
                    q->prior->freq = freq;
                    q = q->prior;
                }
            }
            p = p->next;
        }
    }

    void DispList(LinkNode *L)
    {
        while (L != NULL)
        {
            printf("%d ", L->data);
            L = L->next;
        }

        printf("\n");
    }

    int main()
    {
        int i;
        LinkNode *L = NULL;
        ElemType a[10] = {2, 3, 5, 8, 1, 0, 9, 4, 7, 6};

        InitList(&L, a, 10);
        printf("原链表存储值依次为: ");
        DispList(L);

        printf("请输入访问的值(Ctrl+z to stop): ");
        while((scanf("%d", &i)) != EOF)
            LocateNode(L, i);

        printf("现链表存储值依次为: ");
        DispList(L);
```

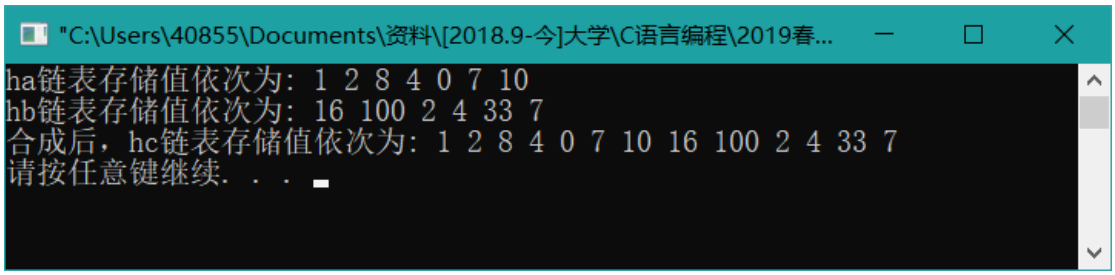
```
        return 0;
    }
```

17、

```
#include<stdio.h>
#include<stdlib.h>

typedef int ElemType;

typedef struct Node
{
    ElemType data;
    struct Node *next;
} LinkNode;

void InitList_ha(LinkNode *&L)
{
    int i;
    int a[7] = {1, 2, 8, 4, 0, 7, 10};
    LinkNode *s, *r;

    L = (LinkNode *)malloc(sizeof(LinkNode));
    r = L;
    for (i = 0; i<7; i++)
    {
        s = (LinkNode *)malloc(sizeof(LinkNode));
```

```
        s->data = a[i];
        r->next = s;
        r = s;
    }
    r->next = L;
}

void InitList_hb(LinkNode *&L)
{
    int i;
    int a[6] = {16, 100, 2, 4, 33, 7};
    LinkNode *s, *r;

    L = (LinkNode *)malloc(sizeof(LinkNode));
    r = L;
    for (i = 0; i<6; i++)
    {
        s = (LinkNode *)malloc(sizeof(LinkNode));
        s->data = a[i];
        r->next = s;
        r = s;
    }
    r->next = L;
}

void Create(LinkNode* ha, LinkNode* hb, LinkNode*& hc)
{
    LinkNode* p = ha->next;
    hc = ha;

    while (p->next != ha)
        p = p->next;

    p->next = hb->next;

    while (p->next != hb)
        p = p->next;

    p->next = hc;
    free(hb);
}

void DispList(LinkNode *L)
{
```

```
    LinkNode *p = L->next;

    while (p != L)
    {
        printf("%d ", p->data);
        p = p->next;
    }

    printf("\n");
}

int main()
{
    int i;
    LinkNode *ha, *hb, *hc;

    InitList_ha(ha);
    InitList_hb(hb);
    printf("ha 链表存储值依次为: ");
    DispList(ha);
    printf("hb 链表存储值依次为: ");
    DispList(hb);

    printf("合成后，hc 链表存储值依次为: ");
    Create(ha, hb, hc);
    DispList(hc);

    return 0;
}
```



第三章

9、

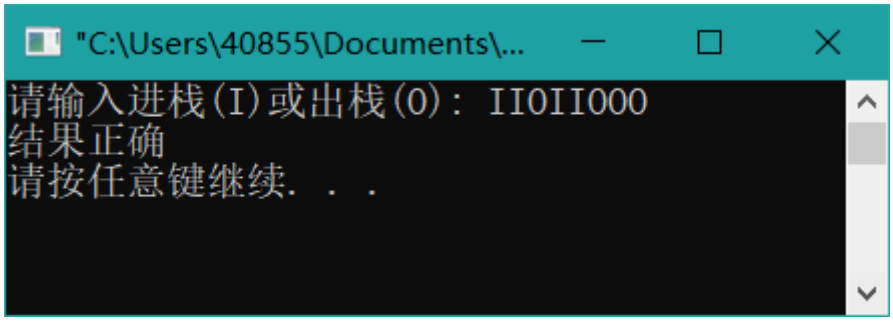
（2）

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
```

```
#define MAXSIZE 20

typedef struct
{
	char data[MAXSIZE];
	int top;
} SqStack;

void InitStack(SqStack*&s){
	s=(SqStack*)malloc(sizeof(SqStack));
	s->top = -1;
}

void DestroyStack(SqStack*&s){
	free(s);
}

bool StackEmpty(SqStack*s){
	return (s->top == -1);
}

bool Push(SqStack*&s, char e)
{
	if(s->top==MAXSIZE-1)
		return false;

	s->top++;
	s->data[s->top] = e;

	return true;
}

bool Pop(SqStack*&s, char &e){
	if(s->top==-1)
		return false;
	 e=s->data[s->top];
	 s->top--;
	 return true;
}

bool Judge(char str[],int n)
{
	int i=0;
	char x;
```

```
    SqStack *ls;
    bool flag=true;
    InitStack(ls);

    while (i<n && flag)
    {
        if (str[i] == 'I')
        {
            Push(ls, str[i]);
        }

        else if (str[i] == 'O')
        {
            if (StackEmpty(ls))
            {
                flag = false;
            }

            else
            {
                Pop(ls,x);
            }
        }

        else
        {
            flag = false;
        }

        i++;
    }

    if (!StackEmpty(ls))
        flag = false;
    DestroyStack(ls);
    return flag;
}

int main(void)
{
    int t = 0;
    char ch, str[MAXSIZE];
    printf("请输入进栈(I)或出栈(O): ");
    while((ch = getchar())!='\n'){
```

```
            str[t] = ch;
            t++;
        }
        str[t] = '\0';

        if(Judge(str, t))
            printf("结果正确\n");
        else
            printf("结果错误\n");

        return 0;

}
```



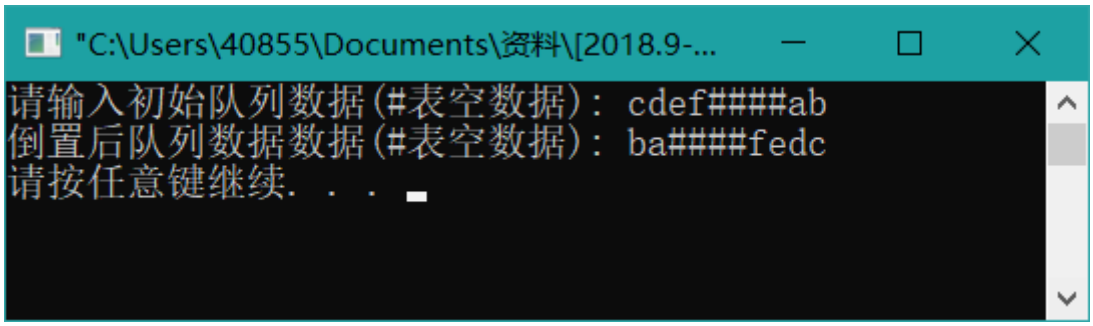
12、

```
#include <stdlib.h>
#include <stdio.h>
#define MAXSIZE 30

typedef struct
{
    char data[MAXSIZE];
    int top;
} SqStack;


typedef struct
{
    char data[MAXSIZE];
    int front, rear;
} SqQueue;

//初始化栈
void InitStack(SqStack*& s)
{
```

```
    s = (SqStack*)malloc(sizeof(SqStack));
    s->top = -1;
}

//销毁栈
void DestroyStack(SqStack*& s)
{
    free(s);
}

//判断空栈
bool StackEmpty(SqStack* s)
{
    return(s->top == -1);
}

//入栈
bool Push(SqStack*& s, char e)
{
    if(s->top == MAXSIZE-1)
        return false;
    s->top++;
     s->data[s->top] = e;
    return true;
}

//出栈
bool Pop(SqStack*& s,char& e)
{
    if(s->top == -1)
         return false;
     e = s->data[s->top];
     s->top--;
     return true;
}

//初始化队列
void InitQueue(SqQueue*& q)
{
    q=(SqQueue*)malloc(sizeof(SqQueue));
    q->front=q->rear=0;
}

//销毁队
```

```
void DestroyQueue(SqQueue*& q)
{
	free(q);
}

//判断空队
bool QueueEmpty(SqQueue*& q)
{
	return(q->front == q->rear);
}

//进队
bool enQueue(SqQueue*& q, char t)
{
	if((q->rear+1) % MAXSIZE == q->front)
		return false;
	q->rear = (q->rear+1) % MAXSIZE;
	q->data[q->rear] = t;
	return true;
}

//出队
bool deQueue(SqQueue*& q, char& t)
{
	if(q->front == q->rear)
		return false;
	q->front = (q->front+1) % MAXSIZE;
	t = q->data[q->front];
	return true;
}

//翻转
void Reverse(SqQueue*& t)
{
	char e;
	SqStack *st;

	InitStack(st);
	while (!QueueEmpty(t))//出队并进栈
	{
		deQueue(t, e);
		Push(st, e);
	}
```

```
    InitQueue(t);
    while (!StackEmpty(st))//出栈并入队
    {
        Pop(st, e);
        enQueue(t, e);
    }
    DestroyStack(st);
}

int main(void)
{
    char ch;
    SqQueue* round;

    InitQueue(round);
    printf("请输入初始队列数据(#表空数据): ");
    while((ch=getchar()) != '\n')
    {
        enQueue(round, ch);
    }

    Reverse(round);

    printf("倒置后队列数据数据(#表空数据): ");
    while(QueueEmpty(round) != 1)
    {
        deQueue(round, ch);
        printf("%c", ch);
    }
    printf("\n");

    return 0;
}
```

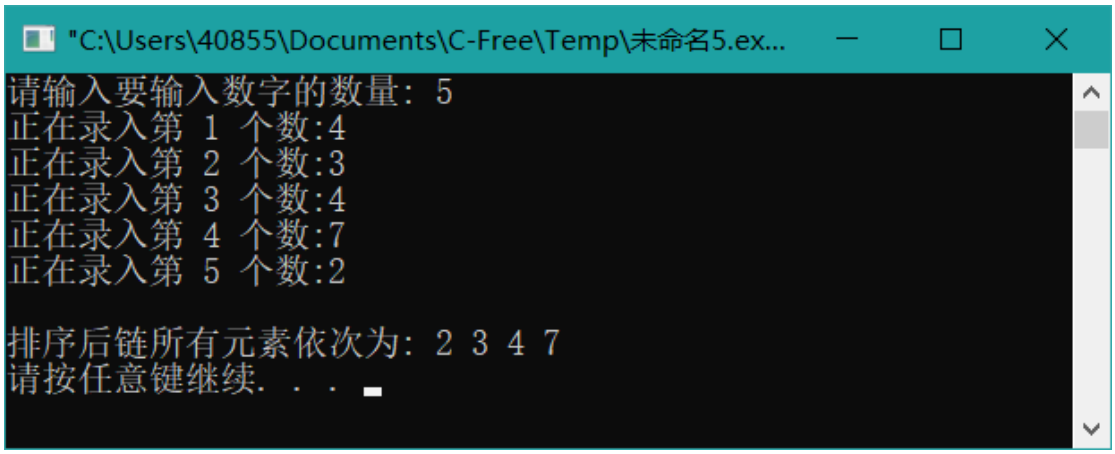


13、

```
#include <stdio.h>
#include <malloc.h>
#define MAXSIZE 10

typedef struct node
{
    int data;
    struct node* next;
} QNode;

void Insert(QNode* OLD[], QNode* NEW[], int x)
{
    QNode* s;
    s = (QNode*)malloc(sizeof(QNode));
    s->data = x;
    s->next = NULL;

    if (OLD[x] == NULL)
    {
        OLD[x] = s;
        NEW[x] = s;
    }

    else
    {
        NEW[x]->next = s;
    }
}

void Create(QNode* OLD[], QNode* NEW[])
{
    int n, x, i;
    printf("请输入要输入数字的数量: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
    {
        do
        {
            printf("正在录入第 %d 个数:", i + 1);
            scanf("%d", &x);
        } while (x < 0 || x>10);
        Insert(OLD, NEW, x);
    }
```

```
}

void DispList(QNode * head)
{
    printf("\n 排序后链所有元素依次为: ");
    while (head != NULL)
    {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

QNode* Link(QNode * OLD[], QNode * NEW[])
{
    QNode* head = NULL, * tail;
    int i;

    for (i = 0; i < MAXSIZE; i++)
        if (OLD[i] != NULL)
        {
            if (head == NULL)
            {
                head = OLD[i];
                tail = NEW[i];
            }
            else
            {
                tail->next = OLD[i];
                tail = NEW[i];
            }
        }

    tail->next = NULL;
    return head;
}

int main()
{
    int i;
    QNode* head;
    QNode* OLD[MAXSIZE], * NEW[MAXSIZE];

    for (i = 0; i < MAXSIZE; i++)
```

```
        OLD[i] = NEW[i] = NULL;

    Create(OLD, NEW);
    head = Link(OLD, NEW);

    DispList(head);
    return 0;
}
```



第四章

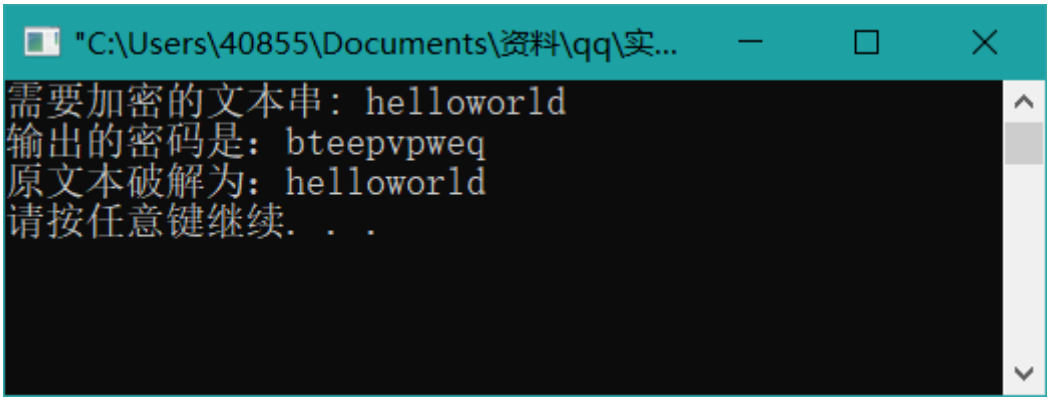
实验 4、

```
#include <stdio.h>
#define MAXSIZE 50

typedef struct
{
    char data[MAXSIZE];
    int length;
} SqString;


void StrAssign(SqString &s, char cstr[])
{
    int i;
    for(i=0; cstr[i]!='\0'; i++)
        s.data[i]=cstr[i];
    s.length=i;
}

void DispStr(SqString s)
{
    int i;
```

```
    if(s.length>0)
    {
        for(i=0; i<s.length; i++)
            printf("%c",s.data[i]);
        printf("\n");
    }
}


int main()
{
    int i=0, j=0;
    char cstr[MAXSIZE], ch;
    SqString cstr_old, cstr_new;

    printf("需要加密的文本串: ");
    while((ch = getchar())!='\n')
    {
        cstr[i]=ch;
        i++;
    }
    cstr[i] = '\0';

    StrAssign(cstr_old, cstr);

     for(j=0; j<cstr_old.length; j++)
    {
        switch(cstr_old.data[j])
        {
            case 'a': cstr_new.data[j]='n';
                    break;
            case 'b': cstr_new.data[j]='g';
                    break;
            case 'c': cstr_new.data[j]='z';
                    break;
            case 'd': cstr_new.data[j]='q';
                    break;
            case 'e': cstr_new.data[j]='t';
                    break;
            case 'f': cstr_new.data[j]='c';
                    break;
            case 'g': cstr_new.data[j]='o';
                    break;
            case 'h': cstr_new.data[j]='b';
```

```
                break;
            case 'i': cstr_new.data[j]='m';
                break;
            case 'j': cstr_new.data[j]='u';
                break;
            case 'k': cstr_new.data[j]='h';
                break;
            case 'l': cstr_new.data[j]='e';
                break;
            case 'm': cstr_new.data[j]='l';
                break;
            case 'n': cstr_new.data[j]='k';
                break;
            case 'o': cstr_new.data[j]='p';
                break;
            case 'p': cstr_new.data[j]='d';
                break;
            case 'q': cstr_new.data[j]='a';
                break;
            case 'r': cstr_new.data[j]='w';
                break;
            case 's': cstr_new.data[j]='x';
                break;
            case 't': cstr_new.data[j]='f';
                break;
            case 'u': cstr_new.data[j]='y';
                break;
            case 'v': cstr_new.data[j]='i';
                break;
            case 'w': cstr_new.data[j]='v';
                break;
            case 'x': cstr_new.data[j]='r';
                break;
            case 'y': cstr_new.data[j]='s';
                break;
            case 'z': cstr_new.data[j]='j';
                break;
        }
    }

    cstr_new.length=cstr_old.length;
    printf("输出的密码是：");
    DispStr(cstr_new);
    printf("原文本破解为：");
```

```
    DispStr(cstr_old);

    return 0;
}
```

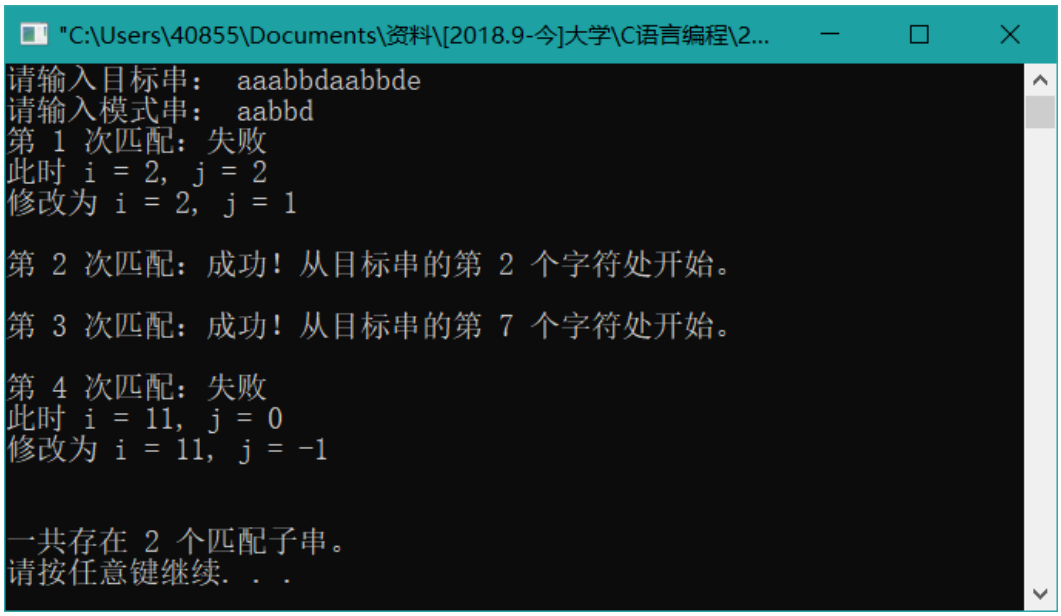


实验 6、

```
#include <stdio.h>
#define MAXSIZE 30

typedef struct{
    char data[MAXSIZE];
    int length;
} SqString;

void StrAssign(SqString &s, char cstr[])
{
    int i;
    for (i=0; cstr[i] != '\0'; i++){
        s.data[i] = cstr[i];
    }
    s.length = i;
}

void GetNext(SqString t, int next[])
{
    int j=0, k=-1;
    next[0] = -1;

    while (j < t.length)
    {
        if (k == -1 || t.data[j] == t.data[k])
        {
            j++; k++;
```

```
                if (t.data[j] != t.data[k])
                {
                    next[j] = k;
                }
                else
                {
                    next[j] = next[k];
                }
            }
            else
            {
                k = next[k];
            }
        }
}

int main(void)
{
    char cstr_t[MAXSIZE], cstr_p[MAXSIZE], ch;
    int i, j, sizet, sizep, next[MAXSIZE], count = 0, num = 0;
    SqString str_t, str_p;

    printf("请输入目标串：  ");
    i = 0;
    while((ch = getchar()) != '\n'){
        cstr_t[i] = ch;
        i++;
    }
    cstr_t[i] = '\0';
    sizet = i;

    printf("请输入模式串：  ");
    i = 0;
    while((ch = getchar()) != '\n'){
        cstr_p[i] = ch;
        i++;
    }
    cstr_p[i] = '\0';
    sizep = i;

    if (sizet < sizep){
        printf("模式串应不长于目标串！\n");
        return 0;
    }
```

```
    StrAssign(str_t, cstr_t);
    StrAssign(str_p, cstr_p);

    //KMP 算法
    i = 0; j = 0;
    GetNext(str_p, next);

    while (i<str_t.length)
    {
        while (j<str_p.length && i<str_t.length)
        {
            if (j==-1 || str_t.data[i]==str_p.data[j])
            {
                i++; j++;
            }
            else
            {
                num++;
                printf("第 %d 次匹配：失败\n 此时 i = %d, j = %d\n", num, i, j);
                j=next[j];
                printf("修改为 i = %d, j = %d\n", i, j);
            }
        }

        if (j>=str_p.length)
        {
            num++;
            printf("第 %d 次匹配:成功!从目标串的第 %d 个字符处开始。\n\n",
num, i-str_p.length+1);
            count++;
        }

        j = 0;
    }

    printf("\n 一共存在 %d 个匹配子串。\n", count);

    return 0;
}
```

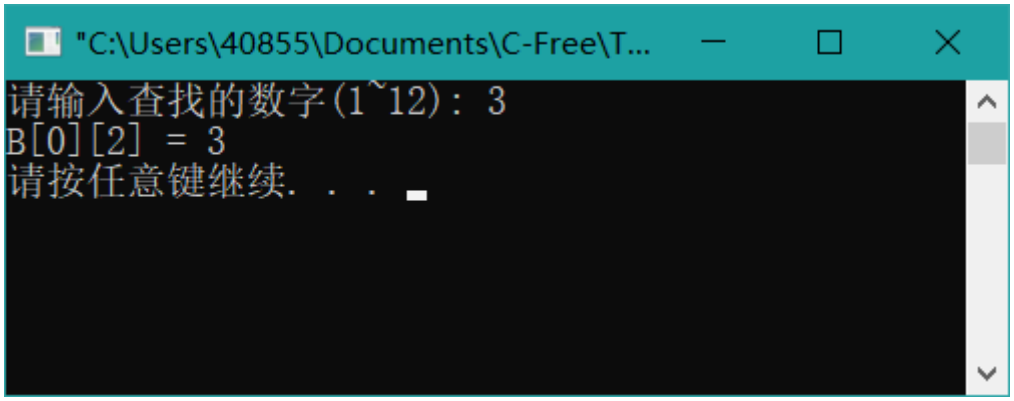


第六章

9、

```
#include <stdio.h>
#define M 3
#define N 4

void Find(int B[M][N], int x, int& i, int& j)
{
    i = 0; j = N - 1;

    while (B[i][j] != x)
    {
        if (B[i][j] > x)
            j--;
        else
            i++;
    }
}

int main()
{
    int i, j, x;
    int B[M][N] = {{1,2,3,4} , {5,6,7,8} , {9,10,11,12}};
    printf("请输入查找的数字(1~12): ");
    scanf("%d", &x);

    if (x < 1 || x > 12)
    {
```

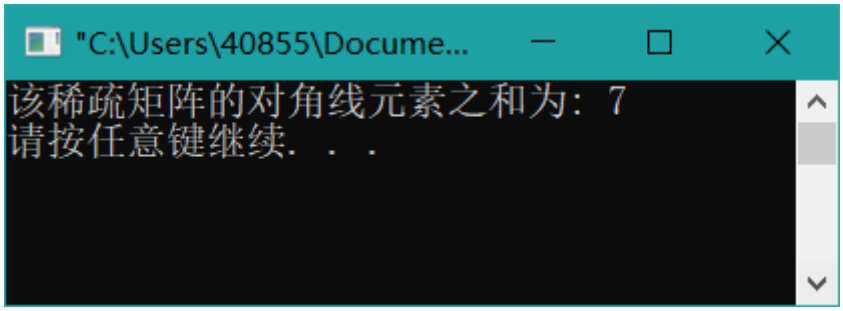
```
        printf("矩阵不存在该数字！\n");
        return -1;
    }

    Find(B, x, i, j);
    printf("B[%d][%d] = %d\n", i, j, x);
    return 0;
}
```



10、

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 16
#define M 4
#define N 4

typedef struct
{
    int row;
    int col;
    int data;
} TupNode;

typedef struct
{
    int rows;
    int cols;
    int nums;
    TupNode datas[MAXSIZE];
} TSMatrix;


void init(TSMatrix &t, int A[M][N])
```

```
{
    int i, j;
    t.rows = M;
    t.cols = N;
    t.nums = 0;

    for (i = 0; i<M; i++)
    {
        for (j = 0; j<N; j++)
        {
            if (A[i][j] != 0)
            {
                t.datas[t.nums].row = i;
                t.datas[t.nums].col = j;
                t.datas[t.nums].data = A[i][j];
                t.nums++;
            }
        }
    }
}

int total(TSMatrix t)
{
    int sum = 0;

    for (int i = 0; i < t.nums; i++)
    {
        if (t.datas[i].row == t.datas[i].col)
            sum += t.datas[i].data;
    }

    return sum;
}

int main(void)
{
    TSMatrix t;
    int A[M][N] = {{1,0,4,5} , {1,0,3,0} , {1,4,3,1} , {0,0,4,3}};
    init(t, A);
    printf("该稀疏矩阵的对角线元素之和为: %d\n", total(t));
    return 0;
}
```

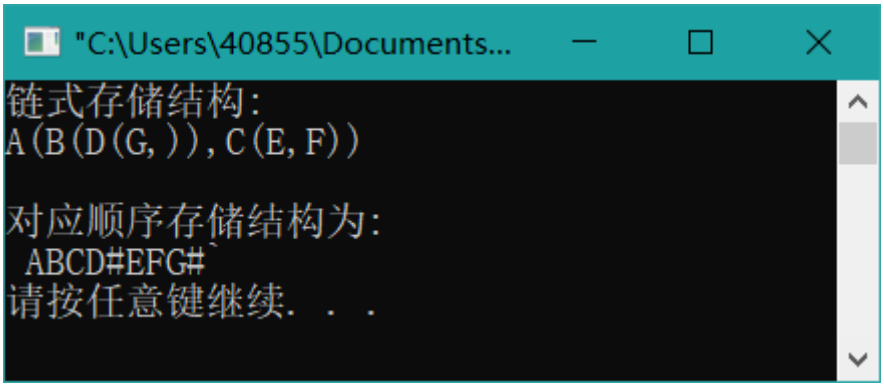


第七章

10、

```
#include <stdio.h>
#include <malloc.h>
#define MAXSIZE 100

typedef char ElemType;
typedef ElemType SqBinTree[MAXSIZE];

typedef struct node
{
    ElemType data;
    struct node *lchild;
    struct node *rchild;
} BTNode;

void CreateBTree(BTNode * &b, char *str){
    BTNode *St[MAXSIZE],*p=NULL;
    int top=-1,k,j=0;
    char ch;
    b=NULL;
    ch=str[j];

    while (ch!='\0')
    {
        switch(ch)
        {
        case '(': top++; St[top]=p; k=1; break;
        case ')': top--; break;
        case ',': k=2; break;
        default: p=(BTNode *)malloc(sizeof(BTNode));
                p->data=ch;p->lchild=p->rchild=NULL;
                if (b==NULL)
                    b=p;
                else
```

```
                    {
                        switch(k)
                        {
                        case 1:St[top]->lchild=p;break;
                        case 2:St[top]->rchild=p;break;
                        }
                    }
            }
            j++;
            ch=str[j];
        }
}

void Trans(BTNode *b, SqBinTree a, int i)
{
    if (b!=NULL)
    {
        a[i]=b->data;
        Trans(b->lchild, a, 2*i);
        Trans(b->rchild, a, 2*i+1);
    }
    else
        a[i]='#';
}

int main(void){
    int i=1;
    BTNode *b;
    SqBinTree a;
    a[0]=' ';
    char str[20]="A(B(D(G,)),C(E,F))";

    CreateBTree(b, str);
    Trans(b, a, i);

    printf("链式存储结构:\n%s\n",str);
    printf("\n");
    printf("对应顺序存储结构为:\n%s\n",a);
}
```

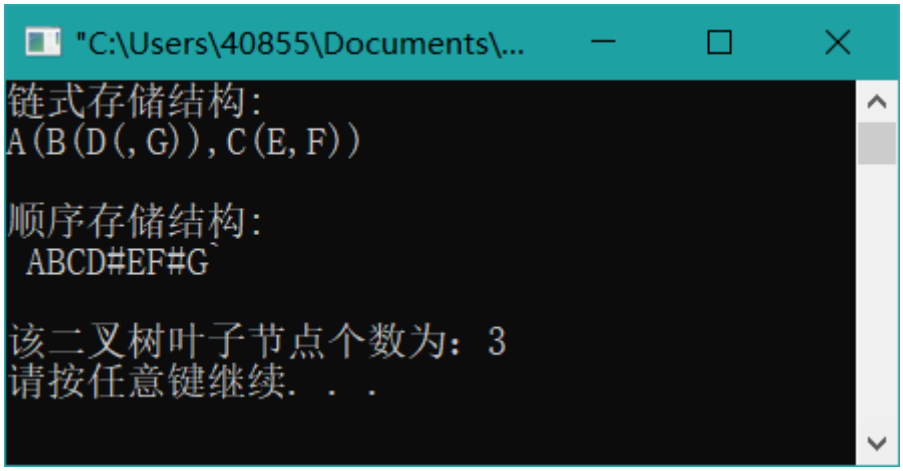


11、

```
#include <stdio.h>
#include <malloc.h>
#define MAXSIZE 100

typedef char ElemType;
typedef ElemType SqBinTree[MAXSIZE];

typedef struct node
{
	ElemType data;
	struct node *lchild;
	struct node *rchild;
} BTNode;

void CreateBTree(BTNode * &b,char *str) {
	BTNode *St[MAXSIZE],*p=NULL;
	int top=-1,k,j=0;
	char ch;
	b=NULL;
	ch=str[j];
	while (ch!='\0')
	{
		switch(ch) {
		case '(':top++;St[top]=p;k=1; break;
		case ')':top--;break;
		case ',':k=2; break;
		default:p=(BTNode *)malloc(sizeof(BTNode));
			p->data=ch;p->lchild=p->rchild=NULL;
			if (b==NULL)
				b=p;
			else	{
				switch(k) {
```

```
                        case 1:St[top]->lchild=p;break;
                        case 2:St[top]->rchild=p;break;
                        }
                    }
            }
            j++;
            ch=str[j];
        }
}
void DestroyBTree(BTNode *&b)
{    if (b!=NULL)
     {    DestroyBTree(b->lchild);
          DestroyBTree(b->rchild);
          free(b);
     }
}

void Translate(BTNode *b,SqBinTree a,int i){
   if (b!=NULL){
      a[i]=b->data;
      Translate(b->lchild,a,2*i);
      Translate(b->rchild,a,2*i+1);
   }
   else a[i]='#';
}

int Nodes(SqBinTree t,int i){
     int numl,numr,num=0;
     if (i<MAXSIZE){
        if (t[i]!='#'){
           if (t[2*i]=='#' && t[2*i+1]=='#')    num++;
           else{
                numl=Nodes(t,2*i);
                numr=Nodes(t,2*i+1);
                num+=numl+numr;
             }
            return num;
          }
          else return 0;
     }
      else return 0;
}

int main(void){
```

```
    int i=1,res;
    BTNode *b;
    SqBinTree a;
    a[0]=' ';
    char str[20]="A(B(D(,G)),C(E,F))";
    CreateBTree(b,str);
    Translate(b,a,i);

    printf("链式存储结构:\n%s\n",str);
    printf("\n");
    printf("顺序存储结构:\n%s\n",a);
    printf("\n");

    i=1;
    res=Nodes(a,i);
    printf("该二叉树叶子节点个数为：%d\n",res);
}
```

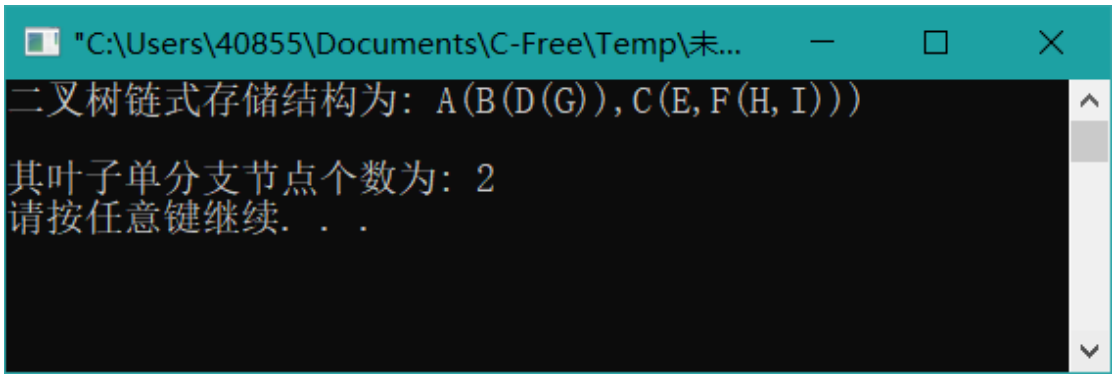


12、

```
#include <stdio.h>
#include <malloc.h>
#define MAXSIZE 100
typedef char ElemType;

typedef struct node
{
    ElemType data;
    struct node *lchild;
    struct node *rchild;
} BTNode;
```

```
void CreateBTree(BTNode * &b,char *str) {
    BTNode *St[MAXSIZE],*p=NULL;
    int top=-1,k,j=0;
    char ch;
    b=NULL;
    ch=str[j];
    while (ch!='\0')
    {
        switch(ch) {
        case '(':top++;St[top]=p;k=1; break;
        case ')':top--;break;
        case ',':k=2; break;
        default:p=(BTNode *)malloc(sizeof(BTNode));
                p->data=ch;p->lchild=p->rchild=NULL;
                if (b==NULL)
                    b=p;
                else   {
                    switch(k) {
                    case 1:St[top]->lchild=p;break;
                    case 2:St[top]->rchild=p;break;
                    }
                }
        }
        j++;
        ch=str[j];
    }
}

int SingleNodes(BTNode *b){
    int numl,numr,counter;

    if (b==NULL)
        return 0;
    else if ((b->lchild==NULL && b->rchild!=NULL) ||(b->lchild!=NULL &&
b->rchild==NULL))
        counter=1;
    else
        counter=0;

    numl=SingleNodes(b->lchild);
    numr=SingleNodes(b->rchild);

    return (numl+numr+counter);
}
```

```
int main(void){
    int i=1,res;
    BTNode *b;
    char str[30]="A(B(D(G)),C(E,F(H,I)))";

    CreateBTree(b,str);
    printf("二叉树链式存储结构为: %s\n\n",str);

    i=1;
    res=SingleNodes(b);
    printf("其叶子单分支节点个数为: %d\n",res);
}
```

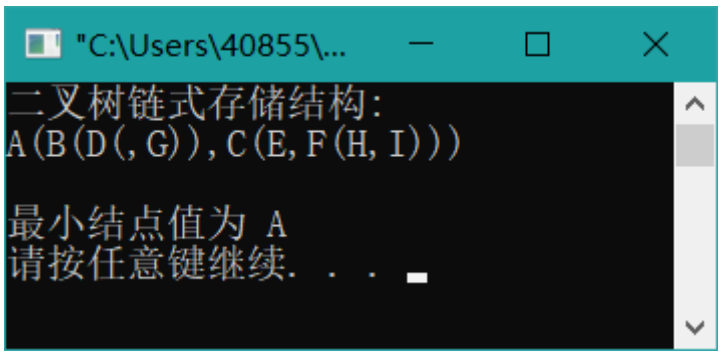


13、

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#define MAXSIZE 100

typedef char ElemType;

typedef struct node
{
    ElemType data;
    struct node *lchild;
    struct node *rchild;
} BTNode;

void CreateBTree(BTNode * &b,char *str)
{
    BTNode *St[MAXSIZE],*p=NULL;
    int top=-1,k,j=0;
    char ch;
```

```
    b=NULL;
    ch=str[j];
    while (ch!='\0')
    {
        switch(ch)
        {
        case '(':top++;St[top]=p;k=1; break;
        case ')':top--;break;
        case ',':k=2; break;
        default:p=(BTNode *)malloc(sizeof(BTNode));
                p->data=ch;p->lchild=p->rchild=NULL;
                if (b==NULL)
                    b=p;
                else
                {
                    switch(k)
                    {
                    case 1:St[top]->lchild=p;break;
                    case 2:St[top]->rchild=p;break;
                    }
                }
        }
        j++;
        ch=str[j];
    }
}
void DestroyBTree(BTNode *&b)
{
    if (b!=NULL)
    {
        DestroyBTree(b->lchild);
        DestroyBTree(b->rchild);
        free(b);
    }
}

void Find (BTNode *b,char &min)
{
    if (b == NULL) return;
     if (b->data<min)    min=b->data;
     Find (b->lchild,min);
     Find (b->rchild,min);
}
```

```
void MinNode(BTNode *b)
{
    if (b!=NULL){
        char min=b->data;
        Find (b,min);
        printf("最小结点值为  %c\n",min);
    }
}

int main(void)
{
    int res;
    BTNode *b;
    char str[30]="A(B(D(,G)),C(E,F(H,I)))";

    CreateBTree(b,str);
    printf("二叉树链式存储结构:\n%s\n",str);
    printf("\n");
    MinNode(b);
}
```

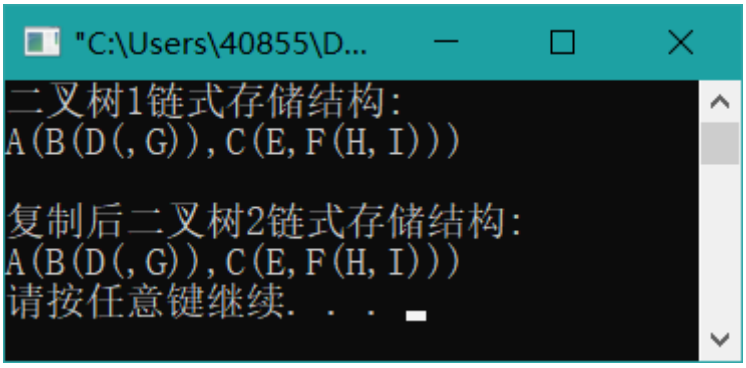


14、

```
#include <stdio.h>
#include <malloc.h>
#define MAXSIZE 100

typedef char ElemType;

typedef struct node
{
    ElemType data;
    struct node *lchild;
    struct node *rchild;
} BTNode;
```

```
void CreateBTree(BTNode * &b,char *str)
{
    BTNode *St[MAXSIZE],*p=NULL;
    int top=-1,k,j=0;
    char ch;
    b=NULL;
    ch=str[j];
    while (ch!='\0')
    {
        switch(ch)
        {
            case '(':top++;St[top]=p;k=1; break;
            case ')':top--;break;
            case ',':k=2; break;
            default:p=(BTNode *)malloc(sizeof(BTNode));
                    p->data=ch;p->lchild=p->rchild=NULL;
                    if (b==NULL)
                        b=p;
                    else
                    {
                        switch(k)
                        {
                        case 1:St[top]->lchild=p;break;
                        case 2:St[top]->rchild=p;break;
                        }
                    }
        }
        j++;
        ch=str[j];
    }
}

void DispBTree(BTNode *b)
{
    if (b!=NULL)
    {
        printf("%c",b->data);
        if (b->lchild!=NULL || b->rchild!=NULL)
        {
            printf("(");
            DispBTree(b->lchild);
            if (b->rchild!=NULL) printf(",");
            DispBTree(b->rchild);
```

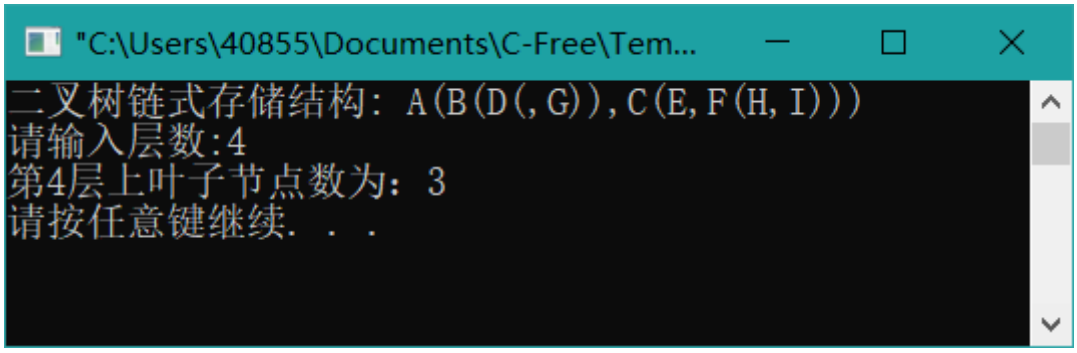
```
                printf(")");
            }
        }
}

void Copy(BTNode *b1,BTNode *&b2)
{
    if (b1==NULL)    b2=NULL;
    else
    {
        b2=(BTNode *)malloc(sizeof(BTNode));
        b2->data=b1->data;
        Copy(b1->lchild,b2->lchild);
        Copy(b1->rchild,b2->rchild);
    }
}

int main(void)
{
    BTNode *b1,*b2;
    char str[30]="A(B(D(,G)),C(E,F(H,I)))";

    CreateBTree(b1,str);

    printf("二叉树 1 链式存储结构:\n%s\n",str);

    Copy(b1,b2);

    printf("\n 复制后二叉树 2 链式存储结构:\n");
    DispBTree(b2);

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



15、

```
#include <stdio.h>
#include <malloc.h>
#define MAXSIZE 100
typedef char ElemType;
typedef struct node
{
    ElemType data;
    struct node *lchild;
    struct node *rchild;
} BTNode;

void CreateBTree(BTNode * &b,char *str)
{
    BTNode *St[MAXSIZE],*p=NULL;
    int top=-1,k,j=0;
    char ch;
    b=NULL;
    ch=str[j];
    while (ch!='\0')
    {
        switch(ch)
        {
        case '(':top++;St[top]=p;k=1; break;
        case ')':top--;break;
        case ',':k=2; break;
        default:p=(BTNode *)malloc(sizeof(BTNode));
                p->data=ch;p->lchild=p->rchild=NULL;
                if (b==NULL)
                    b=p;
                else
                {
                    switch(k)
                    {
                    case 1:St[top]->lchild=p;break;
                    case 2:St[top]->rchild=p;break;
                    }
                }
        }
        j++;
        ch=str[j];
    }
}
```

```
int Count(BTNode *b,int k,int h)
{
        int num1,num2,num=0;
        if (b!=NULL){
                if (h==k && b->lchild==NULL && b->rchild==NULL)    num++;
                num1=Count(b->lchild,k,h+1);
                num2=Count(b->rchild,k,h+1);
                num+=num1+num2;
                return num;
          }
        return 0;
  }

int main(void)
{
      int k, res, i=1;
      BTNode *b;
      char str[30]="A(B(D(,G)),C(E,F(H,I)))";

      CreateBTree(b,str);
      printf("二叉树链式存储结构: %s\n", str);

      printf("请输入层数:");
      scanf("%d",&k);

      res=Count(b,k,i);

      printf("第%d 层上叶子节点数为：%d",k,res);
      printf("\n");
}
```

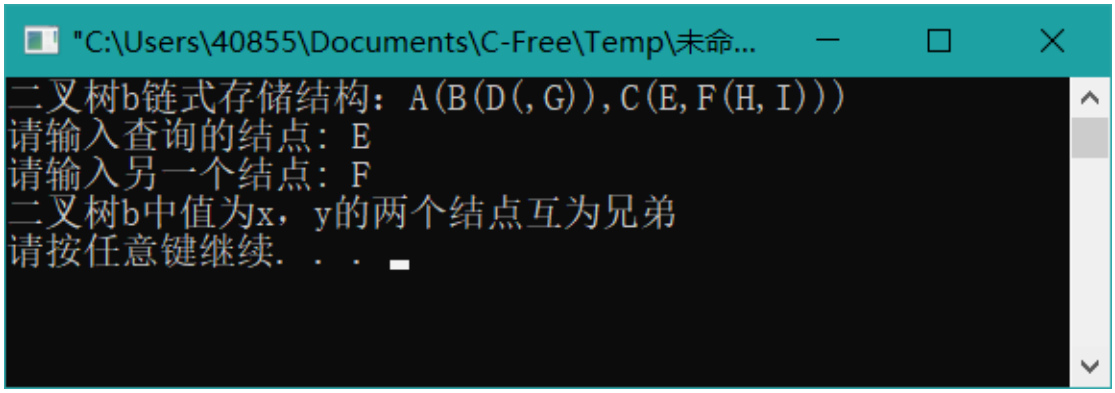


16、

```
#include <stdio.h>
```

```
#include <malloc.h>
#define MAXSIZE 100

typedef char ElemType;

typedef struct node
{
	ElemType data;
	struct node *lchild;
	struct node *rchild;
} BTNode;

void CreateBTree(BTNode * &b,char *str)
{
	BTNode *St[MAXSIZE],*p=NULL;
	int top=-1,k,j=0;
	char ch;
	b=NULL;
	ch=str[j];
	while (ch!='\0')
	{
		switch(ch) {
		case '(':top++;St[top]=p;k=1; break;
		case ')':top--;break;
		case ',':k=2; break;
		default:p=(BTNode *)malloc(sizeof(BTNode));
			p->data=ch;p->lchild=p->rchild=NULL;
			if (b==NULL)
				b=p;
			else	{
				switch(k) {
				case 1:St[top]->lchild=p;break;
				case 2:St[top]->rchild=p;break;
				}
			}
		}
		j++;
		ch=str[j];
	}
}

bool JudgeB(BTNode *b,char x,char y)
{
   bool flag;
```

```
    if (b==NULL)    return false;
    else{
        if (b->lchild!=NULL && b->rchild!=NULL){
            if ((b->lchild->data==x && b->rchild->data==y) ||(b->lchild->data==y &&
b->rchild->data==x))    return true;
        }
        flag=JudgeB(b->lchild,x,y);
        if (flag==true)    return true;
        else      return JudgeB(b->rchild,x,y);
    }
}

int main(void)
{
    int flag=0;
    char x,y;
    BTNode *b;
    char str[30]="A(B(D(,G)),C(E,F(H,I)))";

    CreateBTree(b,str);
    printf("二叉树 b 链式存储结构：%s\n",str);
    printf("请输入查询的结点: ");

    scanf("%c",&x);

    getchar();

    printf("请输入另一个结点: ");
    scanf("%c",&y);

    flag=JudgeB(b,x,y);

    if(flag)
        printf("二叉树 b 中值为 x，y 的两个结点互为兄弟");
    else
        printf ("二叉树 b 中值为 x，y 的两个结点不是兄弟");

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

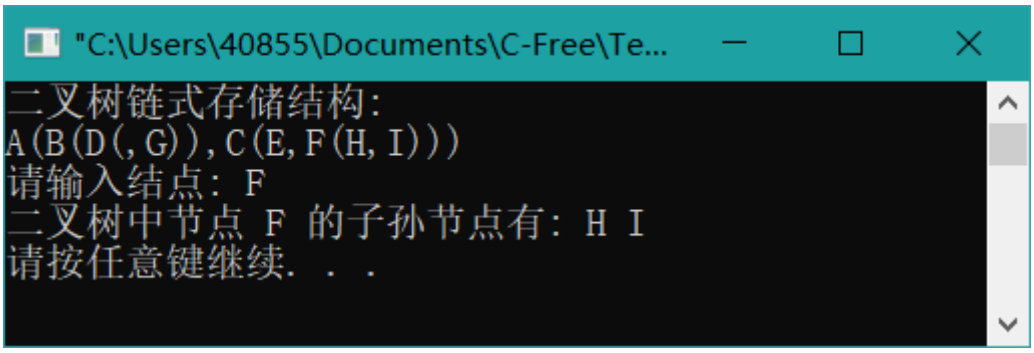


17、

```
#include <stdio.h>
#include <malloc.h>
#define MAXSIZE 100

typedef char ElemType;

typedef struct node
{
    ElemType data;
    struct node *lchild;
    struct node *rchild;
} BTNode;

void CreateBTree(BTNode * &b,char *str)
{
    BTNode *St[MAXSIZE],*p=NULL;
    int top=-1,k,j=0;
    char ch;
    b=NULL;
    ch=str[j];
    while (ch!='\0')
    {
        switch(ch)
        {
        case '(':top++;St[top]=p;k=1; break;
        case ')':top--;break;
        case ',':k=2; break;
        default:p=(BTNode *)malloc(sizeof(BTNode));
                p->data=ch;p->lchild=p->rchild=NULL;
                if (b==NULL)
                    b=p;
                else
```

```
                {
                    switch(k)
                    {
                    case 1:St[top]->lchild=p;break;
                    case 2:St[top]->rchild=p;break;
                    }
                }
            }
            j++;
            ch=str[j];
        }
}

void Output(BTNode *p) {
    if (p!=NULL)
    {
        printf("%c ", p->data);
        Output(p->lchild);
        Output(p->rchild);
    }
}

void Child(BTNode *b,char x)
{
    if (b!=NULL)
    {
        if (b->data==x)
        {
            if (b->lchild!=NULL)
                Output(b->lchild);
            if (b->rchild!=NULL)
                Output(b->rchild);

            return;
        }
        Child(b->lchild, x);
        Child(b->rchild, x);
    }
}

int main(void)
{
    char x;
    BTNode *b;
```

```
    char str[30]="A(B(D(,G)),C(E,F(H,I)))";
    CreateBTree(b,str);

    printf("二叉树链式存储结构:\n%s\n",str);

    printf("请输入结点: ");
    scanf("%c", &x);

    printf("二叉树中节点 %c 的子孙节点有: ", x);
    Child(b, x);

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

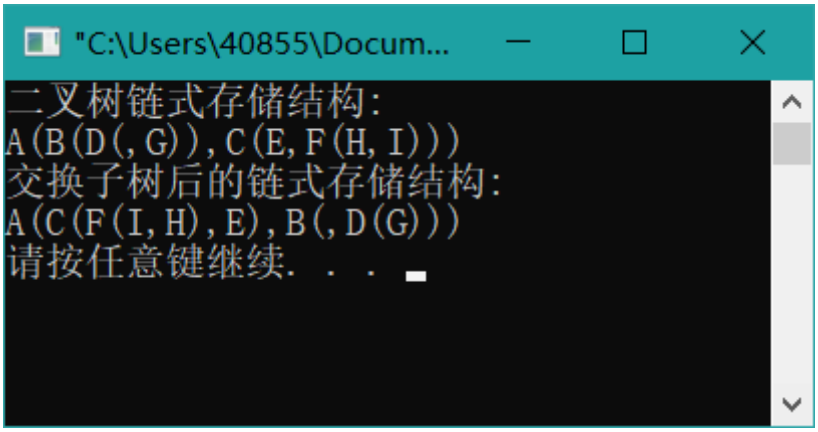


18、

```
#include <stdio.h>
#include <malloc.h>
#define MAXSIZE 100

typedef char ElemType;

typedef struct node
{
    ElemType data;
    struct node *lchild;
    struct node *rchild;
} BTNode;

void CreateBTree(BTNode * &b,char *str)
{
    BTNode *St[MAXSIZE],*p=NULL;
    int top=-1,k,j=0;
    char ch;
```

```
    b=NULL;
    ch=str[j];
    while (ch!='\0')
    {
        switch(ch)
        {
        case '(':top++;St[top]=p;k=1; break;
        case ')':top--;break;
        case ',':k=2; break;
        default:p=(BTNode *)malloc(sizeof(BTNode));
                p->data=ch;p->lchild=p->rchild=NULL;
                if (b==NULL)
                    b=p;
                else
                {
                    switch(k)
                    {
                    case 1:St[top]->lchild=p;break;
                    case 2:St[top]->rchild=p;break;
                    }
                }
        }
        j++;
        ch=str[j];
    }
}

void DispBTree(BTNode *b)
{
    if (b!=NULL)
    {
        printf("%c",b->data);
        if (b->lchild!=NULL || b->rchild!=NULL)
        {
            printf("(");
            DispBTree(b->lchild);
            if (b->rchild!=NULL)
                printf(",");
            DispBTree(b->rchild);
            printf(")");
        }
    }
}
```

```
BTNode *Swap(BTNode *b)
{
    BTNode *t,*tl,*tr;

    if (b==NULL)
        t=NULL;

    else
    {
        t=(BTNode *)malloc(sizeof(BTNode));
        t->data=b->data;
        tl=Swap(b->lchild);
        tr=Swap(b->rchild);
        t->lchild=tr;
        t->rchild=tl;
    }

    return t;
}

int main(void)
{
    char x;
    BTNode *b,*p;
    char str[30]="A(B(D(,G)),C(E,F(H,I)))";
    CreateBTree(b,str);

    printf("二叉树链式存储结构:\n%s\n",str);

    p = Swap(b);

    printf("交换子树后的链式存储结构:\n");
    DispBTree(p);

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

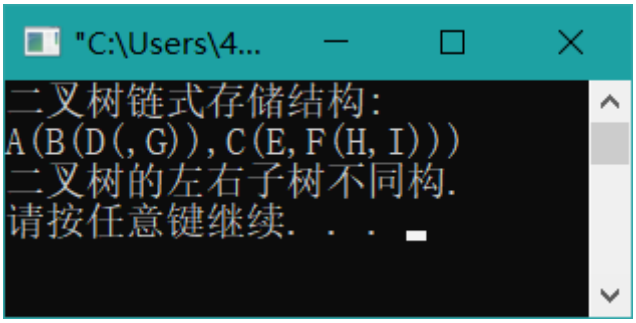


19、

```
#include <stdio.h>
#include <malloc.h>
#define MAXSIZE 100

typedef char ElemType;

typedef struct node
{
    ElemType data;
    struct node *lchild;
    struct node *rchild;
} BTNode;

void CreateBTree(BTNode * &b,char *str)
{
    BTNode *St[MAXSIZE],*p=NULL;
    int top=-1,k,j=0;
    char ch;
    b=NULL;
    ch=str[j];
    while (ch!='\0')
    {
        switch(ch)
        {
        case '(':top++;St[top]=p;k=1; break;
        case ')':top--;break;
        case ',':k=2; break;
        default:p=(BTNode *)malloc(sizeof(BTNode));
                p->data=ch;p->lchild=p->rchild=NULL;
                if (b==NULL)
                    b=p;
```

```
                else
                {
                    switch(k)
                    {
                    case 1:St[top]->lchild=p;break;
                    case 2:St[top]->rchild=p;break;
                    }
                }
            }
            j++;
            ch=str[j];
        }
}

void DispBTree(BTNode *b)
{
    if (b!=NULL)
    {
        printf("%c",b->data);
        if (b->lchild!=NULL || b->rchild!=NULL)
        {
            printf("(");
            DispBTree(b->lchild);
            if (b->rchild!=NULL) printf(",");
            DispBTree(b->rchild);
            printf(")");
        }
    }
}

void DestroyBTree(BTNode *&b)
{   if (b!=NULL)
    {
        DestroyBTree(b->lchild);
        DestroyBTree(b->rchild);
        free(b);
    }
}

bool JudgeChild (BTNode *b1,BTNode *b2)
{
    if (b1==NULL && b2==NULL)
        return true;
    else if (b1==NULL || b2==NULL)
```

```
        return false;
    else
        return (JudgeChild (b1->lchild,b2->lchild) && JudgeChild (b1->rchild,b2->rchild));
}

int main(void)
{
    bool flag;
    BTNode *b;
    char str[30]="A(B(D(,G)),C(E,F(H,I)))";

    CreateBTree(b,str);
    printf("二叉树链式存储结构:\n%s\n",str);

    if (b==NULL)
        flag = true;
     else
        flag = JudgeChild(b->lchild,b->rchild);

     if(flag)
        printf("二叉树的左右子树同构.\n");
    else
        printf ("二叉树的左右子树不同构.\n");

    return 0;
}
```



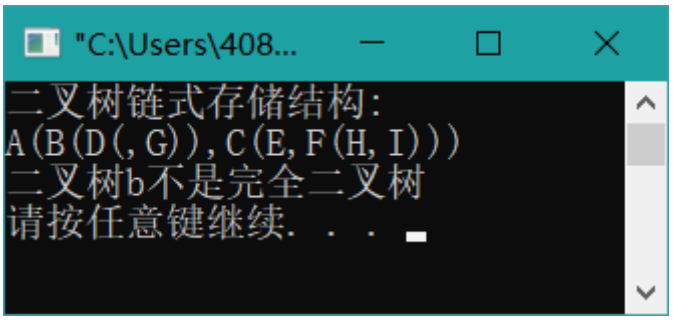
20、

```
#include <stdio.h>
#include <malloc.h>
#define MAXSIZE 100

typedef char ElemType;
```

```
typedef struct node
{
    ElemType data;
    struct node *lchild;
    struct node *rchild;
} BTNode;

void CreateBTree(BTNode * &b,char *str)
{
    BTNode *St[MAXSIZE],*p=NULL;
    int top=-1,k,j=0;
    char ch;
    b=NULL;
    ch=str[j];
    while (ch!='\0')
    {
        switch(ch)
        {
        case '(':top++;St[top]=p;k=1; break;
        case ')':top--;break;
        case ',':k=2; break;
        default:p=(BTNode *)malloc(sizeof(BTNode));
                p->data=ch;p->lchild=p->rchild=NULL;
                if (b==NULL)
                    b=p;
                else
                {
                    switch(k)
                    {
                    case 1:St[top]->lchild=p;break;
                    case 2:St[top]->rchild=p;break;
                    }
                }
        }
        j++;
        ch=str[j];
    }
}

bool CBTree(BTNode *b)
{
    BTNode *Qu[MAXSIZE],*p;
    int front=0,rear=0;
    bool x=true;
```

```
    bool y=true;

    if (b==NULL)
    return true;
    rear++;
    Qu[rear]=b;

    while (front!=rear)
    {
        front=(front+1) % MAXSIZE;
        p=Qu[front];
        if (p->lchild==NULL)
        {
            y=false;
            if (p->rchild!=NULL)
                x=false;
        }
        else
        {
            if (!y)
                x=false;

            rear=(rear+1)%MAXSIZE;
            Qu[rear]=p->lchild;

            if (p->rchild==NULL)
                y=false;
            else
            {
                rear=(rear+1)%MAXSIZE;
                Qu[rear]=p->rchild;
            }
        }
    }
    return x;
}


int main(void){
    bool flag;
    BTNode *b;
    char str[30]="A(B(D(,G)),C(E,F(H,I)))";
    CreateBTree(b,str);
```

```
    printf("二叉树链式存储结构:\n%s\n",str);

     flag=CBTree(b);

     if(flag)
         printf("二叉树 b 是完全二叉树\n");
    else
         printf ("二叉树 b 不是完全二叉树\n");
}
```



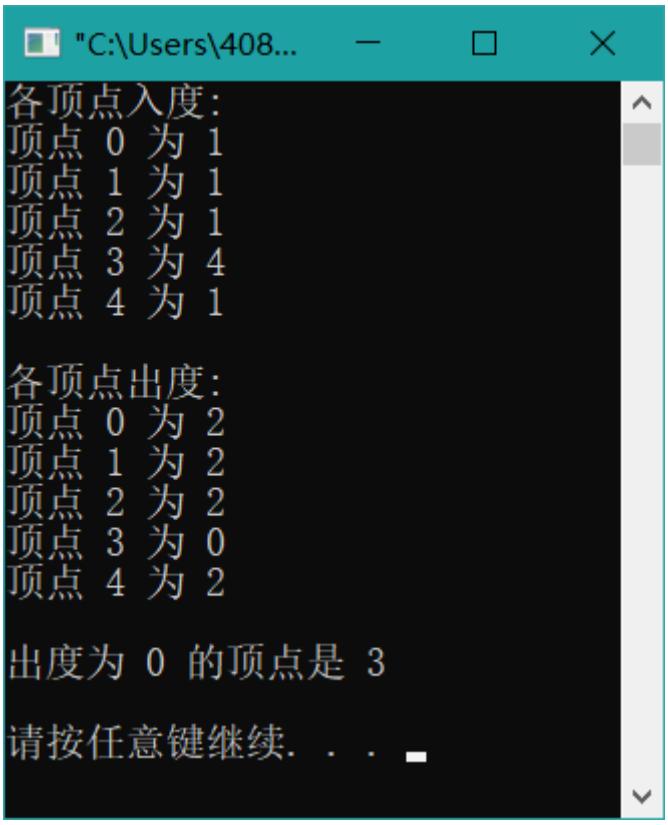
第八章

14、

```
# include <stdio.h>
# include <stdlib.h>
# define MAXSIZE 20

typedef struct
{
    int no;
} VertexType;

typedef struct
{
    int edges[MAXSIZE][MAXSIZE];
    int n, e;
    VertexType vexs[MAXSIZE];
} MatGraph;

void MatGraphCreate(MatGraph& g, int cstr[MAXSIZE][MAXSIZE], int l, int m)
{
    int i = 0, j = 0;
    for (i = 0; i < l; i++)
    {
        for (j = 0; j < m; j++)
        {
```

```
                g.edges[i][j] = cstr[i][j];
            }
        }
        g.n = m;
}

void InDs(MatGraph g)
{
        int i, j, n;
        printf("各顶点入度:\n");

        for (j = 0; j < g.n; j++)
        {
            n = 0;
            for (i = 0; i < g.n; i++)
            {
                if (g.edges[i][j] != 0) n++;
            }
            printf("顶点 %d 为 %d\n", j, n);
        }
}

void OutDs(MatGraph g)
{
        int i, j, n;
        printf("各顶点出度:\n");
        for (i = 0; i < g.n; i++)
        {
            n = 0;
            for (j = 0; j < g.n; j++) {
                if (g.edges[i][j] != 0)
                    n++;
            }
            printf("顶点 %d 为 %d\n", i, n);
        }
}

void ZeroOutDs(MatGraph g)
{
        int i, j, n;
        printf("出度为 0 的顶点是 ");
        for (i = 0; i < g.n; i++)
        {
            n = 0;
```

```
        for (j = 0; j < g.n; j++)
        {
            if (g.edges[i][j] != 0)
                n++;
        }
        if (n == 0)
            printf("%d\n", i);
        else
            printf("不存在");
    }
    printf("\n");
}

int main(void)
{
    int A[MAXSIZE][MAXSIZE] = {{0,1,0,1,0},{0,0,1,1,0},{0,0,0,1,1},{0,0,0,0,0},{1,0,0,1,0}};

    MatGraph g;
    MatGraphCreate(g, A, 5, 5);

    InDs(g);
    printf("\n");
    OutDs(g);
    printf("\n");
    ZeroOutDs(g);
}
```



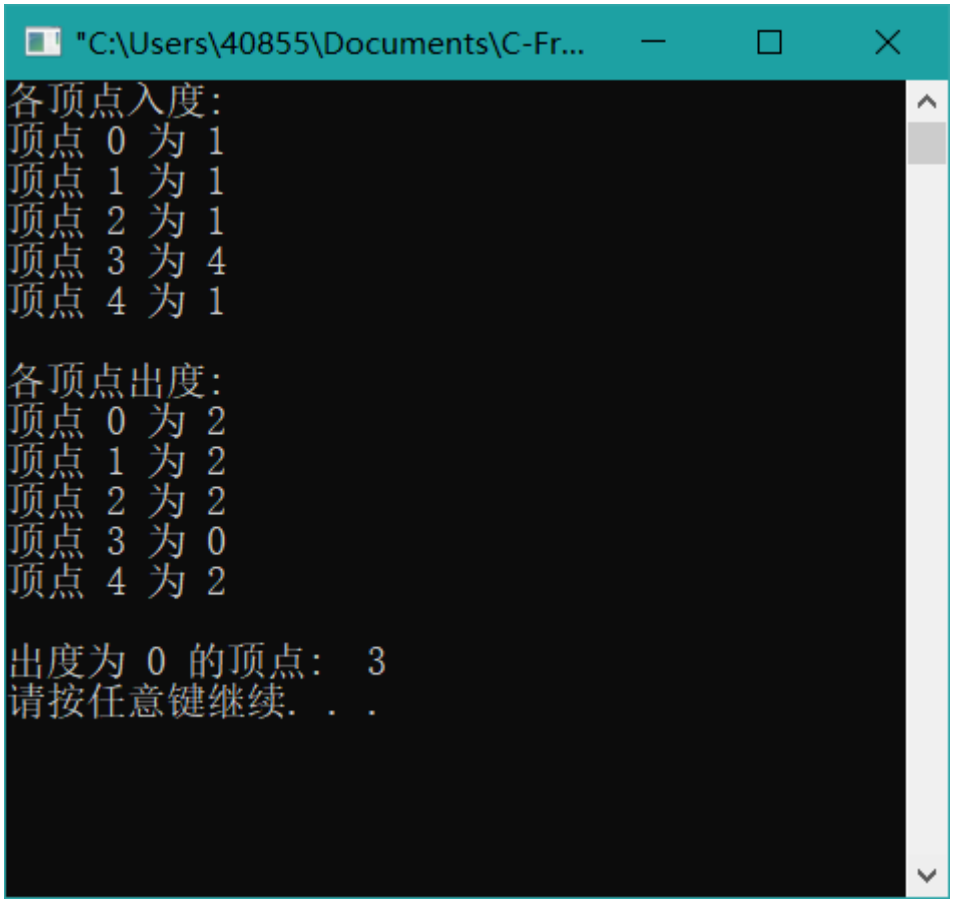
15、

```
# include <stdio.h>
# include <stdlib.h>
# define MAXSIZE 20
# define INF 32767

typedef struct ANode
{
    int adjvex;
    struct ANode* nextarc;
    int weight;
} ArcNode;

typedef struct
{
    ArcNode* firstarc;
} VNode;

typedef struct
{
    VNode adjlist[MAXSIZE];
    int n, e;
} AdjGraph;
```

```
void CreateAdj(AdjGraph*& G, int A[MAXSIZE][MAXSIZE], int n, int e)
{
    int i, j;
    ArcNode* p;
    G = (AdjGraph*)malloc(sizeof(AdjGraph));
    for (i = 0; i < n; i++)
        G->adjlist[i].firstarc = NULL;

    for (i = 0; i < n; i++)
        for (j = n - 1; j >= 0; j--)
            if (A[i][j] != 0 && A[i][j] != INF)
            {
                p = (ArcNode*)malloc(sizeof(ArcNode));
                p->adjvex = j;
                p->weight = A[i][j];
                p->nextarc = G->adjlist[i].firstarc;
                G->adjlist[i].firstarc = p;
            }

    G->n = n;
    G->e = e;
}

void InDs(AdjGraph * G)
{
    ArcNode* p;
    int A[MAXSIZE], i;
    for (i = 0; i < G->n; i++)
        A[i] = 0;
    for (i = 0; i < G->n; i++)
    {
        p = G->adjlist[i].firstarc;
        while (p != NULL)
        {
            A[p->adjvex]++;
            p = p->nextarc;
        }
    }
    printf("各顶点入度:\n");
    for (i = 0; i < G->n; i++)
        printf("顶点 %d 为 %d\n", i, A[i]);
}
```

```
void OutDs(AdjGraph * G)
{
    int i, n;
    ArcNode* p;
    printf("各顶点出度:\n");
    for (i = 0; i < G->n; i++)
    {
        n = 0;
        p = G->adjlist[i].firstarc;
        while (p != NULL)
        {
            n++;
            p = p->nextarc;
        }
        printf("顶点 %d 为 %d\n", i, n);
    }
}

void ZeroOutDs(AdjGraph * G)
{
    int i, n;
    ArcNode* p;
    printf("出度为 0 的顶点: ");
    for (i = 0; i < G->n; i++)
    {
        p = G->adjlist[i].firstarc;
        n = 0;
        while (p != NULL)
        {
            n++;
            p = p->nextarc;
        }
        if (n == 0)
            printf("%2d", i);
    }
    printf("\n");
}

int main(void)
{
    int A[MAXSIZE][MAXSIZE] =
{{0,1,0,1,0},{0,0,1,1,0},{0,0,0,1,1},{0,0,0,0,0},{1,0,0,1,0}};

    AdjGraph* G;
```

```
    CreateAdj(G, A, 5, 8);

    InDs(G);
    printf("\n");
    OutDs(G);

    printf("\n");

    ZeroOutDs(G);
}
```

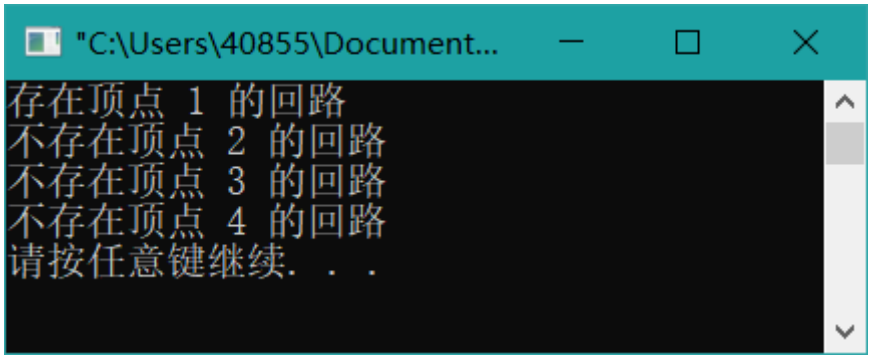


16、

```
# include <stdio.h>
# include <stdlib.h>
# define MAXSIZE 20
# define INF 32767

typedef struct ANode
{
    int adjvex;
    struct ANode* nextarc;
```

```
    int weight;
} ArcNode;

typedef struct
{
    ArcNode* firstarc;
} VNode;

typedef struct
{
    VNode adjlist[MAXSIZE];
    int n, e;
} AdjGraph;

void CreateAdj(AdjGraph*& G, int A[MAXSIZE][MAXSIZE], int n, int e)
{
    int i, j;
    ArcNode* p;
    G = (AdjGraph*)malloc(sizeof(AdjGraph));
    for (i = 0; i < n; i++)
        G->adjlist[i].firstarc = NULL;

    for (i = 0; i < n; i++)
        for (j = n - 1; j >= 0; j--)
            if (A[i][j] != 0 && A[i][j] != INF)
            {
                p = (ArcNode*)malloc(sizeof(ArcNode));
                p->adjvex = j;
                p->weight = A[i][j];
                p->nextarc = G->adjlist[i].firstarc;
                G->adjlist[i].firstarc = p;
            }

    G->n = n;
    G->e = e;
}

int visited[MAXSIZE];

void Cycle(AdjGraph * G, int u, int v, int d, bool& has)
{
    ArcNode* p; int w;
    visited[u] = 1; d++;
    p = G->adjlist[u].firstarc;
```

```
    while (p != NULL)
    {
        w = p->adjvex;
        if (visited[w] == 0)
            Cycle(G, w, v, d, has);
        else if (w == v && d > 1)
        {
            has = true;
            return;
        }
        p = p->nextarc;
    }
}

bool hasCycle(AdjGraph * G, int v)
{
    bool has = false;
    Cycle(G, v, v, -1, has);
    return has;
}

void hasCycleprint(AdjGraph * G, int v)
{
    int i = hasCycle(G, v);
    if (i)
        printf("存在顶点 %d 的回路\n", v);
    else
        printf("不存在顶点 %d 的回路\n", v);
}

int main(void)
{
    int A[MAXSIZE][MAXSIZE] =
{{0,1,0,1,0},{0,0,1,1,0},{0,0,0,1,1},{0,0,0,0,0},{1,0,0,1,0}};

    AdjGraph* G;
    CreateAdj(G, A, 5, 8);

    hasCycleprint(G, 1);
    hasCycleprint(G, 2);
    hasCycleprint(G, 3);
    hasCycleprint(G, 4);
}
```



17、

```
# include <stdio.h>
# include <stdlib.h>
# define MAXSIZE 20
# define INF 32767

typedef struct ANode
{
    int adjvex;
    struct ANode* nextarc;
    int weight;
} ArcNode;

typedef struct
{
    ArcNode* firstarc;
} VNode;

typedef struct
{
    VNode adjlist[MAXSIZE];
    int n, e;
} AdjGraph;

void CreateAdj(AdjGraph*& G, int A[MAXSIZE][MAXSIZE], int n, int e)
{
    int i, j;
    ArcNode* p;
    G = (AdjGraph*)malloc(sizeof(AdjGraph));
    for (i = 0; i < n; i++)
        G->adjlist[i].firstarc = NULL;

    for (i = 0; i < n; i++)
        for (j = n - 1; j >= 0; j--)
```

```
            if (A[i][j] != 0 && A[i][j] != INF)
            {
                p = (ArcNode*)malloc(sizeof(ArcNode));
                p->adjvex = j;
                p->weight = A[i][j];
                p->nextarc = G->adjlist[i].firstarc;
                G->adjlist[i].firstarc = p;
            }

    G->n = n;
    G->e = e;
}

int visited[MAXSIZE];

void DFS(AdjGraph * G, int v, int& vn, int& en)
{
    ArcNode* p;
    visited[v] = 1; vn++;
    p = G->adjlist[v].firstarc;
    while (p != NULL)
    {
        en++;
        if (visited[p->adjvex] == 0)
            DFS(G, p->adjvex, vn, en);
        p = p->nextarc;
    }
}

int IsTree(AdjGraph * G)
{
    int vn = 0, en = 0, i;
    for (i = 0; i < G->n; i++)
        visited[i] = 0;
    DFS(G, 1, vn, en);
    if (vn == G->n && en == 2 * (G->n - 1))
        return 1;
    else
        return 0;
}

void IsTreeprint(AdjGraph * G)
{
    int i = IsTree(G);
```

```
    if (i)
        printf("验证的图是树\n");
    else
        printf("验证的图不是树\n");
}

int main(void)
{
    int A[MAXSIZE][MAXSIZE] = {{0,1,0,0,1},{1,0,1,0,0},{0,1,0,1,0},{0,0,1,0,1},{1,0,0,1,0}};

    AdjGraph* G;
    CreateAdj(G, A, 5, 5);

    IsTreeprint(G);
}
```