## 第五章 数组和广义表

一、选择题

1. 常对数组进行的两种基本操作是（ ）

（A）建立与删除（B）索引和修改（C）查找和修改（D）查找与索引

2.二维数组 M 的元素是 4 个字符(每个字符占一个存储单元)组成的串,行下标 i 的范围从 0 到 4,列下标 j 的范围从 0 到 5,M 按行存储时元素 M[3][5]的起始地址与 M 按列存储时元素( ）的起始地址相同。

（A）M[2][4]（B）M[3][4]（C）M[3][5]（D）M[4][4]

3.数组 A[8][10]中,每个元素 A 的长度为 3 个字节,从首地址 SA 开始连续存放在存储器内,存放该数组至少需要的单元数是（ )。

（A）80（B）100（C）240（D）270

4.数组 A[8][10]中,每个元素 A 的长度为 3 个字节,从首地址 SA 开始连续存放在存储器内,该数组按行存放时,元素 A[7][4]的起始地址为（ )。

（A）SA+141（B）SA+144（C）SA+222（D）SA+225

5.数组 A[8][10]中,每个元素 A 的长度为 3 个字节,从首地址 SA 开始连续存放在存储器内,该数组按列存放时,元素 A[4][7]的起始地址为（ )。

（A）SA+141（B）SA+180（C）SA+222（D）SA+225

6.稀疏矩阵一般的压缩存储方法有两种,即（ )。

（A） 二维数组和三维数组（B）三元组和散列

（C）三元组和十字链表 （D）散列和十字链表

7.若采用三元组压缩技术存储稀疏矩阵,只要把每个元素的行下标和列下标互换,就完成了对该矩阵的转置运算,这种观点（ )。

（A）正确（B）错误

8.设矩阵 A 是一个对称矩阵,为了节省存储,将其下三角部分按行序存放在一维数组 B[1,n(n-1)/2]中,对下三角部分中任一元素 ai,j(i<=j),在一组数组 B 的下标位置 k 的值是（　）。

（A）i(i-1)/2+j-1（B）i(i-1)/2+j（C）i(i+1)/2+j-1 (D)i(i+1)/2+j

**二、填空题**

1.己知二维数组 A[m][n]采用行序为主方式存储,每个元素占 k 个存储单元,并且第一个元素的存储地址是 LOC(A[0][0]),则 A[0][0]的地址是_____。

2.二维数组 A[10][20]采用列序为主方式存储,每个元素占一个存储单元,并且 A[0][0]的存储地址是 200,则 A[6][12]的地址是_____。

3.有一个 10 阶对称矩阵 A,采用压缩存储方式(以行序为主,且 A[0][0]=1),则 A[8][5]的地址是_____。

4.设 n 行 n 列的下三角矩阵 A 已压缩到一维数组 S[1..n*(n+1)/2]中,若按行序为主存储,则 A[i][j]对应的 S 中的存储位置是_____。

5.若 A 是按列序为主序进行存储的 4×6 的二维数组,其每个元素占用 3 个存储单元,并且 A[0][0]的存储地址为 1000,元素 A[1][3]的存储地址为____,该数组共占用_____个存储单元。

**三、算法设计**

1.如果矩阵 A 中存在这样的一个元素 A[i][j]满足条件:A[i][j]是第 i 行中值最小的元素,且又是第 j 列中值最大的元素,则称之为该矩阵的一个马鞍点。编写一个函数计算出 1×n 的矩阵 A 的所有马鞍点。

2.n 只猴子要选大王,选举办法如下:所有猴子按 1,2,...,n 编号围坐一圈,从 1

号开始按 1、2、…、m 报数,凡报 m 号的退出到圈外,如此循环报数,直到圈内剩下只猴子时,这只猴子就是大王。n 和 m 由键盘输入,打印出最后剩下的猴子号。编写一程序实现上述函数。

3.数组和广义表的算法验证程序

编写下列程序:

(1)求广义表表头和表尾的函数 head()和 tail()。

(2)计算广义表原子结点个数的函数 count_GL()。

(3)计算广义表所有原子结点数据域(设数据域为整型)之和的函数 sum_GL()。

**参考答案：**

**一、选择题**

1. C 2.B 3. C 4. C 5. B 6.C 7、B 8、B

**二、填空题**

1、loc(A[0][0])+(n*i+j)*k 2、332 3、42 4、i*(i+1)/2+j+1 5、1039；72

**三、算法设计题**

1.

算法思想：依题意,先求出每行的最小值元素,放入 min[m]之中,再求出每列的最大值元素,放入 max[n]之中,若某元素既在 min[i]中,又在 max[j]中,则该元素 A[i][j]便是马鞍点,找出所有这样的元素,即找到了所有马鞍点。因此,实现本题功能的程序如下:

```
#include <stdio.h>
#define m 3
#define n 4
void minmax(int a[m][n])
{
int i1,j,have=0;
int min[m],max[n];
for(i1=0;i1<m;i1++)/*计算出每行的最小值元素,放入 min[m]之中*/
{
```

```
min[i1]=a[i1][0];
for(j=1;j<n;j++)
if(a[i1][j]<min[i1]) min[i1]=a[i1][j];
}
for(j=0;j<n;j++)/*计算出每列的最大值元素,放入 max[n]之中*/
{
max[j]=a[0][j];
for(i1=1;i1<m;i1++)
if(a[i1][j]>max [j]) max[j]=a[i1][j];
}
for(i1=0;i1<m;i1++)
for(j=0;j<n;j++)
if(min[i1]==max[j])
{
printf("(%d,%d):%d\n",i1,j,a[i1][j]);
have=1;
}
if(!have) printf("没有鞍点 \ n");
}
```

2.

算法思想:本题用一个含有 n 个元素的数组 a,初始时 a[i]中存放猴子的编号 i,计数器似的值为 0。从 a[i]开始循环报数,每报一次,计数器的值加 1,凡报到 m

时便打印出 a[i]值(退出圈外的猴子的编号),同时将 a[i]的值改为 O(以后它不再参加报数),计数器值重新置为 0。该函数一直进行到 n 只猴子全部退出圈外为止,最后退出的猴子就是大王。因此,现本题功能的程序如下:

```
#include "stdio.h"
main()
{
int a[100];
int count,d,j,m,n;
scanf("%d %d",&m,&n);/* n>=m*/
for(j=0;j<n;j++)
a[j]=j+1;
count=0;d=0;
while(d<n)
for(j=0;j<n;j++)
if(a[j]!=0)
{
count++;
if(count==m)
{
printf("% d ",a[j]);
a[j]=0;
count=0;
```

```
d++;
}
}
}
```

3.

```
#include "stdio.h"
#include "malloc.h"
typedef struct node
{ int tag;
union
{struct node *sublist;
char data;
}dd;
struct node *link;
}NODE;
NODE *creat_GL(char **s)
{
NODE *h;
char ch;
ch=*(*s);
(*s)++;
if(ch!='\0')
```

```
{
h=(NODE*)malloc(sizeof(NODE));
if(ch=='(')
{
h->tag=1;
h->dd.sublist=creat_GL(s);
}
Else
{
h->tag=0;
h->dd.data=ch;
}
}
else
h=NULL;
ch=*(*s);
(*s)++;
if(h!=NULL)
if(ch==',')
h->link =creat_GL(s);
else
h->link=NULL;
```

```
return(h);
}
void prn_GL(NODE *p)
{
if(p!=NULL)
{
if(p->tag==1)
{
printf("(");
if(p->dd.sublist ==NULL)
printf(" ");
else
prn_GL(p->dd.sublist );
}
else
printf("%c",p->dd.data);
if(p->tag==1)
printf(")");
if(p->link!=NULL)
{
printf(",");
prn_GL(p->link);
```

```
}
}
}
NODE *copy_GL(NODE *p)
{
NODE *q;
if(p==NULL) return(NULL);
q=(NODE *)malloc(sizeof(NODE));
q->tag=p->tag;
if(p->tag)
q->dd.sublist =copy_GL(p->dd.sublist );
else
q->dd.data =p->dd.data;
q->link=copy_GL(p->link);
return(q);
}
NODE *head(NODE *p) /*求表头函数 */
{
return(p->dd.sublist);
}
NODE *tail(NODE *p) /*求表尾函数 */
{
```

```
return(p->link);
}
int sum(NODE *p) /*求原子结点的数据域之和函数 */
{ int m,n;
if(p==NULL) return(0);
else
{ if(p->tag==0) n=p->dd.data;
else
n=sum(p->dd.sublist);
if(p->link!=NULL)
m=sum(p->link);
else m=0;
return(n+m);
}
}
int depth(NODE *p) /*求表的深度函数 */
{
int h,maxdh;
NODE *q;
if(p->tag==0) return(0);
else
if(p->tag==1&&p->dd.sublist==NULL) return 1;
```

```
else
{
maxdh=0;
while(p!=NULL)
{
if(p->tag==0) h=0;
else
{q=p->dd.sublist;
h=depth(q);
}
if(h>maxdh)maxdh=h;
p=p->link;
}
return(maxdh+1);
}
}
main()
{
NODE *hd,*hc;
char s[100],*p;
p=gets(s);
hd=creat_GL(&p);
```

```
prn_GL(head(hd));

prn_GL(tail(hd));

hc=copy_GL(hd);

printf("copy after:");

prn_GL(hc);

printf("sum:%d\n",sum(hd));

printf("depth:%d\n",depth(hd));

}
```