



西北工业大学  
NORTHWESTERN POLYTECHNICAL UNIVERSITY

---

# C程序设计 Programming in C



**1011014**

---

主讲：姜学锋，计算机学院

# 使用动态内存

## 1、动态内存的应用

### 7.6.3 动态内存的应用

---

- ▶ 虽然动态内存分配适用于所有数据类型，但通常用于数组、字符串、字符串数组、自定义类型及复杂数据结构类型。

### 7.6.3 动态内存的应用

---

- ▶ 动态内存不同于静态内存，在实际编程中，需要注意以下几点：
- ▶ （1）静态内存管理由编译器进行，程序员只做对象定义（相当于分配），而动态内存管理按程序员人为的指令进行。

### 7.6.3 动态内存的应用

---

- ▶ (2) 动态内存分配和释放必须对应，即有分配就必须有释放，不释放内存会产生“内存泄漏”，后果是随着程序运行多次，可以使用的内存空间越来越少；另一方面，再次释放已经释放的内存空间，会导致程序出现崩溃性错误。

### 7.6.3 动态内存的应用

---

- ▶ (3) 静态分配内存的生命期由编译器自动确定，要么是程序运行期，要么是函数执行期。动态分配内存的生命期由程序员决定，即从分配时开始，至释放时结束。特别地，动态分配内存的生命期允许跨多个函数。

### 7.6.3 动态内存的应用

---

- ▶ (4) 静态分配内存的对象有初始化，动态分配内存一般需要人为的指令赋初始值。
- ▶ (5) 避免释放内存后出现“迷途指针”，应及时设置为空指针。

### 7.6.3 动态内存的应用

---



#### 【例7.25】

---

在不同函数中分配、使用、释放动态内存。



## 7.6.3 动态内存的应用

例7.25

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int *f1(int n)
4 { //分配n个整型内存, 返回首地址
5     int *p, i;
6     p = (int *)malloc(n*sizeof(int)); //分配
7     for (i=0; i<n; i++) p[i]=i; //赋初始值
8     return p;
9 }
10 void f2(int *p, int n)
11 { //输出动态内存中的n个数据
12     while (n-->0) printf("%d ", *p++);
13 }
14 void f3(int *p)
15 { //释放内存
```

### 7.6.3 动态内存的应用

例7.25

```
16  free(p);
17  }
18  int main()
19  {
20      int *pi;
21      pi=f1(5); //分配
22      f2(pi,5); //输出
23      f3(pi); //释放
24      return 0;
25  }
```

### 7.6.3 动态内存的应用

例7.25

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int *f1(int n)
4 { //分配n个整型内存, 返回首地址
5     int *p, i;
6     p = (int *)malloc(n*sizeof(int)); //分配
7     for (i=0; i<n; i++) p[i]=i; //赋初始值
8     return p;
9 }
```

第6行分配n个整型内存单元，第7行给分配到的每个内存单元赋初始值。

### 7.6.3 动态内存的应用

---

例7.25

```
18 int main()  
19 {  
20     int *pi;  
21     pi=f1(5); //分配  
22     f2(pi,5); //输出  
23     f3(pi); //释放  
24     return 0;  
25 }
```

### 7.6.3 动态内存的应用

---

- ▶ 1. 动态分配数组
- ▶ 使用动态内存，可以轻而易举地解决本节开始提出的问题：在程序运行时产生任意大小的“数组”。

### 7.6.3 动态内存的应用

---

- ▶ 动态分配一维或多维数组的方法是由指针管理数组，二维以上数组按一维数组方式来处理，具体步骤为：
- ▶ ①定义指针p；
- ▶ ②分配数组空间，用来存储数组元素，空间大小按元素个数计算；
- ▶ ③按一维数组方式使用这个数组（例如输入、输出等）；
- ▶ 若是一维数组，则元素为 $p[i]$ ；若是二维数组，则元素为 $p[i*M+j]$ ，其中M为列元素个数，以此类推。
- ▶ ④释放数组空间；

### 7.6.3 动态内存的应用

---



#### 【例7.26】

---

计算 $n$ 阶行列式的值（ $n$ 由键盘输入）。

### 7.6.3 动态内存的应用

例7.26

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 double HLS(double *A,int N) //HLS(double A[N][N],int N)
4 { //计算N阶行列式
5     int i,j,m,n,s,t,k=1;
6     double f=1.0, c, x;
7     for (i=0,j=0;i<N && j<N; i++,j++) {
8         if (A[i*N+j]==0) { //A[i][j] 检查主对角线是否为0
9             for (m=i+1; m<N && A[m*N+j]==0; m++); //A[m][j]
10            if (m==N) return 0; //全为0则行列式为0
11        } else
12            for (n=j;n<N;n++) { //两行交换
13                c=A[i*N+n]; //A[i][n]
14                A[i*N+n]=A[m*N+n]; //A[i][n]=A[m][n];
15                A[m*N+n]=c; //A[m][n]
```



### 7.6.3 动态内存的应用

例7.26

```
16         }
17         k=-k;
18     }
19     for (s=N-1;s>i;s--) { //列变换化成上三角行列式
20         x=A[s*N+j]; //A[s][j]
21         for (t=j;t<N;t++) //A[s][t]-=A[i][t]*(x/A[i][j])
22             A[s*N+t]-=A[i*N+t]*(x/A[i*N+j]);
23     }
24 }
25 for (i=0;i<N;i++) f*=A[i*N+i]; //A[i][i]
26 return k*f;
27 }
28 int main()
29 {
30     int i,j,n=4;
```


### 7.6.3 动态内存的应用

例7.26

```
31  double *A;  
32  scanf("%d",&n);  
33  A=malloc(n*n*sizeof(double)); //分配“数组”A[n][n]  
34  for (i=0;i<n;i++)  
35      for (j=0;j<n;j++)  
36          scanf("%lf",A+i*n+j); //输入数据到A[i][j]  
37      printf ("detA=%lf\n",HLS(A,n));  
38  free(A); //释放“数组”  
39  return 0;  
40 }
```

## 7.6.3 动态内存的应用

例7.26



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     double *p;
6     p = (double *)malloc(10 * sizeof(double));
7     if (p == NULL)
8     {
9         printf("Memory allocation failed!\n");
10        return 1;
11    }
12    double detA = 40.000000;
13    printf("detA = %f\n", detA);
14    for (int i = 0; i < 10; i++)
15    {
16        p[i] = 3.14159265358979323846;
17    }
18    for (int i = 0; i < 10; i++)
19    {
20        printf("%f ", p[i]);
21        if (i % 5 == 4)
22            printf("\n");
23    }
24    return 0;
25 }
```

detA=40.000000

4 ✓  
3 1 -1 2 -5 1 3 -4 2 0 1 -1 1 -5 3 -3 ✓

### 7.6.3 动态内存的应用

---

- ▶ 2. 动态分配字符串
- ▶ 实际编程中，字符串类型表示文字信息数据，其特点是字符串长度不固定。通过动态分配字符串，根据程序的需要确定字符串的实际长度。

### 7.6.3 动态内存的应用

---

- ▶ 动态分配字符串的方法是由字符指针管理字符串，具体步骤为：
  - ▶ ①定义字符指针；
  - ▶ ②分配字符串空间，用来存储字符串；
  - ▶ ③使用这个字符串（例如输入、输出等）；
  - ▶ ④释放字符串空间；

### 7.6.3 动态内存的应用

---

► 例如:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     char *p; //字符串指针
6     p=(char*)malloc(1000*sizeof(char)); //分配字符串空间
7     gets(p); //输入字符串
8     puts(p); //输出字符串
9     free(p); //释放字符串空间
10    return 0;
11 }
```

### 7.6.3 动态内存的应用

---

- ▶ 3. 动态分配字符串数组
- ▶ 使用二维字符数组来存储字符串可能会浪费内存空间。采用指针数组和动态内存分配，可以存储多个字符串而且减少不必要的内存开销。

### 7.6.3 动态内存的应用

---

- ▶ 动态分配字符串数组的方法是由指向字符指针的指针管理多个字符指针，由每个字符指针管理字符串，具体步骤为：
- ▶ ①定义字符指针数组的指针（即指向字符指针的指针）；
- ▶ ②分配字符指针数组空间，用来存储若干字符串的指针；
- ▶ ③分配字符串空间，用来存储字符串；
- ▶ ④使用这些字符串（例如输入、输出等）；
- ▶ ⑤释放字符串空间；
- ▶ ⑥释放字符指针数组空间。



### 7.6.3 动态内存的应用

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     int i, n;
6     char **pp;
7     scanf("%d", &n); //输入字符串数目
8     pp=(char**)malloc(n*sizeof(char*)); //分配字符指针数组空间
9     for (i=0; i<n; i++) {
10         pp[i]=(char*)malloc(100*sizeof(char)); //分配字符串空间
11         gets(pp[i]); //输入字符串
12     }
13     for (i=0; i<n; i++) puts(pp[i]); //输出字符串
14     for (i=0; i<n; i++) free(pp[i]); //释放字符串空间
15     free(pp); //释放字符指针数组空间
16     return 0;
```

### 7.6.3 动态内存的应用

---



#### 【动态内存应用举例】

---

编程输出 $n$ 阶魔方阵（ $n$ 由键盘输入）。

### 7.6.3 动态内存的应用



#### 例题分析

根据前面述及的 $n$ 阶魔方阵算法：

- (1) 奇数( $n=2K+1$ )阶魔方阵；
- (2) 双偶( $n=4K$ )阶魔方阵；
- (3) 单偶( $n=4K+2$ )阶魔方阵；

分别设计3个函数给予实现，且使用动态内存而非数组实现“任意的 $n$ 阶”。

### 7.6.3 动态内存的应用

例7.51

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void OutMagic(int **A,int n)
4 { //输出n阶魔方阵
5     int i,j,L,S;
6     for(i=0; i<n; i++) { //输出魔方阵
7         S=0; //S累计一行之和
8         printf(" ");
9         for(j=0; j<n; S=S+A[i][j],j++)
10             printf("%4d ", A[i][j]);
11         printf(" =%4d\n",S); //输出一行之和
12     }
13     printf("=");
14     for(j=0; j<n; j++) { //输出魔方阵列之和
15         S=0;
```

### 7.6.3 动态内存的应用

例7.51

```
16     for(i=0; i<n; i++) S=S+A[i][j];
17     printf("%4d ",S); //输出一列之和
18 }
19 L=S=0;
20 for(i=0; i<n; i++) S=S+A[i][i] , L=L+A[i][n-1-i];
21 printf("  =%4d  =%4d\n",S,L); //输出对角线与反对角线之和
22 }
23 void Magic2k1(int **A,int n,int idx,int ox,int oy)
24 { //奇数(n=2K+1)阶魔方阵,从idx开始填写
25     int i, x, y=0;
26     x=n/2;
27     for(i=idx+1; i<=idx+n*n; i++) {
28         A[oy+y][ox+x]=i;
29         if(i%n==0) y++;
30         else x++,y--;
```

### 7.6.3 动态内存的应用

例7.51

```
31      x=(x%n+n)%n;
32      y=(y%n+n)%n;
33  }
34 }
35 void Magic4k(int **A,int n)
36 { //双偶(n=4K)阶魔方阵
37     int i, j, m, L, S;
38     int x=0, y=0, ox, oy;
39     L=n*n;
40     for(i=1; i<=L; i++) { //先填数字
41         A[y][x]=i;
42         if(i%n==0) x=0, y++;
43         else x++;
44     }
45     S=1+L; //最大数和最小数之和
```

### 7.6.3 动态内存的应用

例7.51

```
46  m=n/4; //以4x4大小分割魔方阵
47  x=y=ox=oy=0;
48  L=m*m;
49  for(j=1; j<=L; j++) {
50      for(i=0;i<4;i++) { //对每个4x4魔方阵做对角互补替换
51          A[oy+i][ox+i]=S-A[oy+i][ox+i];
52          A[oy+i][ox+(4-i-1)]=S-A[oy+i][ox+(4-i-1)];
53      }
54      if(j%m==0) ox=0, oy+=4;
55      else ox=j%m*4; //转移到下一个4x4魔方阵
56  }
57 }
58 void Magic4k2(int **A,int n)
59 { //单偶(n=4K+2)阶魔方阵
60     int i, j, k, m, L;
```

### 7.6.3 动态内存的应用

例7.51

```
61 Magic2k1(A,n/2,0,0,0); //填写A象限
62 Magic2k1(A,n/2,n*n/4,n/2,n/2); //填写D象限
63 Magic2k1(A,n/2,n*n/2,n/2,0); //填写B象限
64 Magic2k1(A,n/2,n*n*3/4,0,n/2); //填写C象限
65 m=(n-2)/4; //对ADBC象限奇数阶魔方阵做处理
66 for(i=0;i<n/2;i++) {
67     for(j=0;j<m;j++) { //步骤2
68         k=(i==n/4)?n/4+j:j;
69         L=A[i][k], A[i][k]=A[i+n/2][k], A[i+n/2][k]=L;
70     }
71     for(j=0;j<m-1;j++) { //步骤3
72         k=n/2+n/4+j;
73         L=A[i][k], A[i][k]=A[i+n/2][k], A[i+n/2][k]=L;
74     }
75 }
```



## 7.6.3 动态内存的应用

例7.51

```
76 }
77 int main()
78 { //求解任意n阶魔方阵
79     int **A, i, n;
80     scanf("%d",&n); //输入n阶
81     if (n>2) {
82         A = (int**)malloc(sizeof(int*)*n); //分配n个数组行指针
83         for(i=0;i<n;i++)
84             A[i]=(int*)malloc(sizeof(int)*n); //分配行
85         if (n%2==1)
86             Magic2k1(A,n,0,0,0); //奇数阶魔方阵
87         else if (n%4==0)
88             Magic4k(A,n); //双偶阶魔方阵
89         else
90             Magic4k2(A,n); //单偶阶魔方阵
```

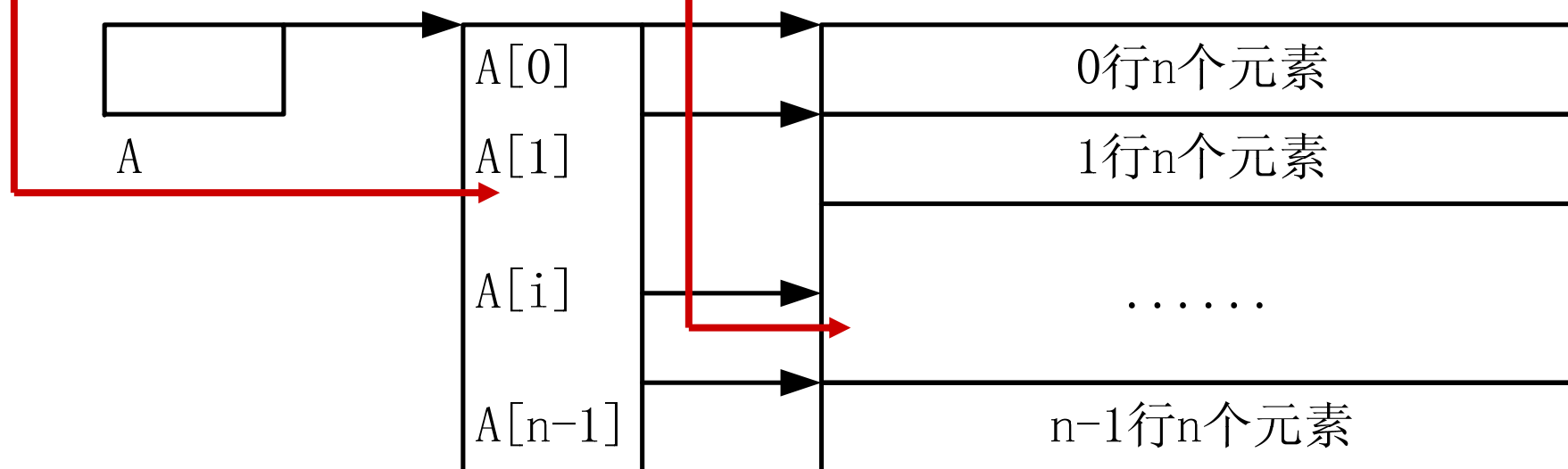
### 7.6.3 动态内存的应用

例7.51

```
91      OutMagic(A,n); //输出魔方阵
92      for(i=0;i<n;i++)
93          free(A[i]); //释放行
94      free(A); //释放行指针
95  }
96  else printf("error input: n>2\n");
97  return 0;
98 }
```

### 7.6.3 动态内存的应用

```
77 int main()  
78 { //求解任意n阶魔方阵  
79     int **A, i, n;  
80     scanf("%d",&n); //输入n阶  
81     if (n>2) {  
82         A = (int**)malloc(sizeof(int*)*n); //分配n个数组行指针  
83         for(i=0;i<n;i++)  
84             A[i]=(int*)malloc(sizeof(int)*n); //分配行
```



**CP** 程序设计