二、高维数组和指针

- 1. 动态存储分配
- 2. 二维数组的定义和二维数组元素的引用
- 3. 二维数组和指针
- 4. 二维数组名和指针数组作为实参
- 5. 高维数组（高阶张量）

重点内容

1. 动态存储分配

(1.1) malloc函数

函数的调用形式： malloc(size)

其中，malloc函数返回值的类型为void *；size的类型为unsigned int。

功能：分配size个字节的存储区,返回一个指向存储区首地址的基类型为void的地址。若没有足够的内存单元供分配,函数返回空(NULL)。

在动态申请存储空间时,若不能确定数据类型所占字节数,可以使用**sizeof运算符**来求得。

例如:

```
pi =( int *) malloc ( sizeof (int) );
pf =( float *) malloc ( sizeof (float) );
```

可以用free函数释放。

例:
设有以下语句:

```
double *p;
 p = _____malloc( sizeof(double) );
```

若要使指针p指向一个double 类型的动态存储单元，在下划线填入

A) double　B) double *　C) (* double)　D) (double *)

1. 动态存储分配

(1.1) malloc函数

➢存储分配函数 malloc()

函数原型 void* malloc(size_t n)


```
int n;
double* scores;
scanf( “%d” ,&n);
scores = (double*)malloc(n*sizeof(double));
if(scores == NULL){ … }
for (int i = 0; i < n; i++) scores[i] = rand()*1.0/RAND_MAX;
```

1. 动态存储分配

(1.2) calloc函数

函数的调用形式为:

calloc (n, size);

其中：n和size的类型都为unsigned int，calloc函数返回值的类型为**void ***。

功能：**给n个同一类型的数据项分配连续的存储空间**,每个数据项的长度为size个字节。若分配成功,函数返回存储空间的首地址;否则返回空。通过调用该函数所分配的存储单元,系统自动置初值0。

(1.3) free函数

函数的调用形式为:

free(p);

这里指针变量p必须指向由动态分配函数malloc或calloc分配的地址。

功能：将指针p所指的存储空间释放。此函数没有返回值。


```
char *ps;
ps =( char *) calloc (10, sizeof (char) );
```

以上函数调用语句开辟了10个连续的char类型的存储单元,由ps指向存储单元的首地址。

使用calloc函数开辟的动态存储单元,可以用free函数释放。

1. 动态存储分配

(1.2) calloc函数 和 (1.3) free函数

➢存储释放函数 free()

函数原型　void free(void *p)


```
int n;
double* scores;
scanf( “%d” ,&n);
scores = (double*)calloc(n, sizeof(double));
if(scores == NULL){ … }
for (int i = 0; i < n; i++) scores[i] = rand()*1.0/RAND_MAX;
  … //其它代码
free(scores);
```

2. 二维数组的定义和二维数组元素的引用

2.1 二维数组和数组元素的地址

先给出以下定义:

int *p, a[3][4];

1） 二维数组a由若干个一维数组组成

二维数组实际上是一个一维数组, 这个一维数组的每个元素又是一个一维数组。如以上定义的a数组, 则可视a数组由a[0]、a[1]、a[2]三个元素组成, 而a[0]、a[1]、a[2]中每个元素又是由四个整型元素组成的一维数组。可用a[0][0]、a[0][1]等来引用a[0]中的每个元素, 其他以此类推。

在二维数组中, a[0]、a[1]、a[2]都是一维数组名, 代表一个不可变的地址常量, 其值依次为二维数组每行第一个元素的地址, 其基类型就是数组元素的类型。

a[0] +1中1的单位应当是2个字节。

p =a[i]; 是合法的， 也可写成: p =*(a +i);

2. 二维数组的定义和二维数组元素的引用

2.1 二维数组和数组元素的地址

2）二维数组名也是一个地址值常量

二维数组名也是一个存放地址常量的指针，其值为二维数组中第一个元素的地址。
数组名a的值与a[0]的值相同，只是其基**类型**为具有**4个整型元素的数组类型**。即
a+0的值与a[0]的值相同
a+1的值与a[1]的值相同
a+2的值与a[2]的值相同
它们分别表示a数组中第一、第二、第三行的首地址。

p =a; 不合法， a++, a =a +i 不合法

2. 二维数组的定义和二维数组元素的引用

2.1 二维数组和数组元素的地址

例1： 已知二维数组 double a[2][2]={ {1, 2}, {3, 4}}; 请回答下列问题

（1）a[0], a[1]分别指什么？a[2] 是否有意义？

（2）若存在某函数 void function(int N, double *a)； 在调用后函数function(2, a);后

在函数function内部的a[2]是否有意义？如果有，指什么？用printf("%f",a[2])查看，回答为什么这样？

(3) a由什么构成？理解为什么a赋初值时要用{ {1, 2}, {3, 4}}这种格式？

如果定义一个数组b[3][2], 对其赋初值格式该如何？

2. 二维数组的定义和二维数组元素的引用

2.1 二维数组和数组元素的地址

3) 二**维数组元素的地址**与**数组元素的引用**

a[i][j]的地址可用以下五种表达式求得:
(1) &a[i][j]
(2) a[i] + j
(3) * (a +i) + j
(4) &a[0][0] +4 * i +j　/*在i行前尚有4*i个元素存在*/
(5) a[0] +4 * i +j

则a数组元素可用以下五种表达式来引用:

1) a[i][j]
2) *(a[i] + j)
3) *(*(a +i) + j)
4) (*(a +i))[j]
5) *(&a[0][0] +4* i +j)

3. 二维数组和指针

3.1 通过建立一个**指针数组**引用二维数组元素

double a[2][3]={{1,2,3},{4,5,6}};
double *p[2] = {a[0], a[1]}; //**这里*p[2] 是一个指针数组**

a数组元素a[i][j]的引用形式*(a[i] + j)和*(p[i] + j)是完全等价的。

通过指针数组p来引用a数组元素的等价形式有:

1) *(p[i] + j) /*与*(a[i] + j)对应*/
2) *(*(p +i) + j) /*与*(*(a +i) + j)对应*/
3) (*(p +i))[j] /*与(*(a +i))[j]对应*/
4) p[i][j] /*与a [i][j]对应*/

区别: p[i]中的值是可变的,而a[i]中的值是不可变的。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    double a[2][3]={{1,2,3},{4,5,6}};
    double *p[2] = {a[0], a[1]};
    for (int i=0; i<2; i++)
      for(int j=0; j<3; j++)
        printf("%f ",p[i][j]);
   return 0;
}
```

3. 二维数组和指针

3.2 通过建立一个**行指针**引用二维数组元素

若有以下定义:

int a[3][2], (* prt)[2];

在说明符(* prt)[2]中, *号首先与prt结合,说明prt是一个**指针变量**,然后再与说明符[2]结合,说明指针变量prt的基类型是**一个包含有两个int元素的数组**。在这里, prt的基类型与a的相同。

prt =a;　合法

prt +1　等价于a +1、a[1]

当prt指向a数组的开头时,可以通过以下形式来引用a[i][j]:

1) *(prt[i] + j)　/*与*(a[i] + j)对应*/

2) *(*(prt +i) + j)　/*与*(*(a +i) + j)对应*/

3) (*(prt +i))[j]　/*与(*(a +i))[j]对应*/

4) prt [i][j]　/*与a [i][j]对应*/

在这里, prt是个指针变量,它的值可变,而a是一个常量。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a[3][2]={{1,2},{3,4},{5,6}};
    int (*prt)[2]; // prt是一个行指针
    prt = a;
    for (int i=0; i<3; i++)
       for(int j=0; j<2; j++)
          printf("%d ",prt[i][j]);
   return 0;
}
```

3. 二维数组和指针

例：指针数组和行指针的区别

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a[3][2]={{1,2},{3,4},{5,6}};
    int (*prt)[2]; // prt是一个行指针
    prt = a;
   int *p[3]; // p 是一个指针数组,
   p[0] = a[0]; p[1] = a[1]; p[2] = a[2]; //不能使用p = a;  为什么?
    for (int i=0; i<3; i++)
       for(int j=0; j<2; j++)
          printf("%d ",prt[i][j]);
    printf("\n");
    for (int i=0; i<3; i++)
       for(int j=0; j<2; j++)
          printf("%d ",p[i][j]);
   return 0;
}
```

4. 二维数组名和指针数组作为实参

4.1 二维数组名作为实参时实参和形参之间的数据传递

当二维数组名作为实参时,对应的形参必须是一个行指针变量。例如,若主函数中有以下定义和函数调用语句:

```
#include <stdio.h>
#define M 5
#define N 3
main ( )
{  double s[M][N];
        …
  fun ( s );
        …
}
```

则fun函数的首部可以是以下三种形式之一:

1) fun (double (* a)[N])
2) fun (double a[ ][N])
3) fun (double a[M][N])

因为在二维数组中, a[i] 都是一维数组名, 代表**一个不可变的地址常量**, 其值依次为二维数组每行第一个元素的地址, 所以系统都将把a处理成**一个行指针**，其列下标不可缺。和一维数组相同, 数组名传送给函数的是一个地址值, 因此, 对应的形参也必定是一个类型相同的**指针变量**, 在函数中引用的将是主函数中的数组元素,系统只为形参开辟一个存放地址的存储单元。

4. 二维数组名和指针数组作为实参

例：编写程序，通过调用随机函数给5×6的二维数组元素赋10~40范围内的整数,求出二维数组每行元素的平均值。

程序设计思想：1、模块化，肢解，单独函数完成单独模块功能。2、然后或者先主函数。

（1）主函数

```
#include <stdio.h>
#include <stdlib.h>
#define M 6
#define N 5
void getdata( int (*)[M] );
void lineave ( int [ ][M], float * );
void outdata ( int [N][M], float * );
main( )
{  int r[N][M];
  float ave[N];
  getdata( r );
  lineave( r, ave );
  outdata( r, ave );
}
```

（2）调用随机函数给5×6的二维数组元素赋10~40范围内的整数

```
void getdata( int (* sp)[M] )
{  int i, j, x;
  for ( i =0; i<N; i++)
  {  j =0;
    while ( j<M )
    {  x =rand()%41;
      if ( x >=10 )
      {  sp[i][j] = x; j++; }
    }
  }
}
```

（3）二维数组每行元素的平均值

```
void lineave ( int s[ ][M], float *a )
{  int i, j;
float ave;
  for ( i =0; i<N; i++)
  {  ave =0.0;
   for ( j =0; j<M; j ++) ave =ave+ s[i][j];
   a[i] = ave/M;
  }
}
```

4. 二维数组名和指针数组作为实参

“当二维数组名作为实参时,对应的形参必须是一个行指针类型变量”。

这个限制条件是为了传递过去后仍以二维数组来使用！比如上页例题函数中的**sp[i][j] 和 s[i][j]。**

这种在函数中维持二维数组原形式必须引入指针数组或行指针，对初学者和一般码员，容易出错!

为此，我们定义函数时统一**采用指针作为形参**，避开指针数组或行指针，同时在函数内部我们使用动态内存分配函数malloc(n*sizeof())， calloc(n, sizeof()) 和free()相结合的方式来分配内存（地址）。这就是我们强调用**指针作为形参**的原因，而且用**指针作为形参**可以很方便推广到**核心的高维数组（高阶张量**，见后面的课件内容）。

```
fun ( int M, int N, double *A)
{
  double *B;
  B=calloc(M*N, sizeof(double));
   ┆
  free(B);
}
```

例2：写出交换矩阵任意两行的函数 、交换矩阵任意两列的函数

```
#include <stdio.h>
#include <stdlib.h>
void arrout(int M,int N,double*A);
void rowchange(int N,double*B, int
row1, int row2);
void colchange(int M,int N,double*C,
int col1,int col2);
int main()
{
    double a[3][2]={{1,2},{3,4},{5,6}};
    double b[3][2]={{1,2},{3,4},{5,6}};
    double c[3][2]={{1,2},{3,4},{5,6}};
    printf("a= \n");
    arrout(3,2,a);
    rowchange(2,b,0,2);
    printf("b= \n");
    arrout(3,2,b);
    colchange(3,2,c,0,1);
    printf("c= \n");
    arrout(3,2,c);
   return 0;
}
```

```
void arrout(int M, int N, double*A)
{
   for(int i=0; i<M; i++)
      for(int j=0; j<N; j++)
        {
         printf("%f ", A[i*N+j]);
         if(j==N-1) printf("\n");
        }
}
```

```
void rowchange(int N,double*B,int
row1,int row2)
{
   double s;
   for(int col=0; col<N; col++)
     {
      s=B[row1*N+col];
      B[row1*N+col]=B[row2*N+col];
      B[row2*N+col]=s;
     }
}
```

```
void colchange(int M,int N,double*C,
int col1,int col2)
{
   double s;
   for(int row=0; row<M; row++)
     {
      s=C[row*N+col1];
      C[row*N+col1]=C[row*N+col2];
      C[row*N+col2]=s;
     }
}
```

例3：编写一个函数来求二阶张量和一阶张量的指标缩并$\sum_j A_{ij}B_j = C_i$

```
void contract(int M,int N,double*A,double*B,double*C)
{
    for(int row=0;row<M;row++)
      {
       double sum = 0;
       for(int col=0;col<N;col++)
           sum += A[row*N+col]*B[col];
       C[row] = sum;
      }
}
```

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

void contract(int M,int N,double*A,double*B,double*C);
int main()
{
    double a[3][2]={{1,2},{3,4},{5,6}};
    double b[2]={7,8};
    double c[3];
    contract(3,2,a,b,c);
    printf("c=\n");
    for(int i=0;i<3;i++)
      printf("%f \n",c[i]);
    return 0;
}
```

例4：编写一个函数来求两个矩阵（具有相同行或列）的合并

```
void rowcombine(int M1,int M2,int
N,double*A,double*B,double*D)
{
    for(int i=0; i<M1+M2;i++)
        for(int j=0;j<N;j++)
         {
           if(i<M1) D[i*N+j]=A[i*N+j];
           else D[i*N+j]=B[(i-M1)*N+j];
         }
}
```

```
void colcombine(int M,int N1,int
N2,double*A,double*C,double*F)
{
    int N = N1+N2;
    for(int i=0; i<M; i++)
        for(int j=0; j<N;j++)
         {
          if (j<N1) F[i*N+j]=A[i*N1+j];
          else F[i*N+j]=C[i*N2+j-N1];
         }
}
```

```
#include <stdio.h>
#include <stdlib.h>
void arrout(int M,int N,double*A);
void rowcombine(int M1,int M2,int N,double*A,double*B,double*D);
void colcombine(int M,int N1,int N2,double*A,double*C,double*F);
int main()
{
    double a[3][2]={{1,2},{3,4},{5,6}};
    double b[2][2]={{7,8},{9,10}};
    double c[3][3]={{11,12,13},{14,15,16},{17,18,19}};
    double d[5][2];    double f[3][5];
    printf("a=\n");     arrout(3,2,a);
    printf("b=\n");     arrout(2,2,b);
    printf("c=\n");     arrout(3,3,c);
    rowcombine(3,2,2,a,b,d);
    printf("d=\n");     arrout(5,2,d);
    colcombine(3,2,3,a,c,f);
    printf("f=\n");     arrout(3,5,f);
    return 0;
}
```

```
void arrout(int M, int N, double*A)
{
    for(int i=0; i<M; i++)
        for(int j=0; j<N; j++)
          {
           printf("%f ", A[i*N+j]);
           if(j==N-1) printf("\n");
          }
}
```

5. 高维数组（高阶张量）：存储机制

```
#include <stdio.h>
#include <stdlib.h>
#define M 2
#define N 3
#define K 4
int main( )
{
    double cube[M][N][K] =
    {
        { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} },
        { {13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23, 24} }
    };
    double *p;
    p = cube;
    for(int i =0; i<24; i++) printf("%d ",*(p+i));
    printf("\n");
    for(int m=0; m<M; m++)
        for(int n=0; n<N; n++)
          for(int k=0; k<K; k++)
            printf("a[%d][%d][%d] = %d \n", m, n, k, cube[m][n][k]);
}
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

```
a[0][0][0] = 1
a[0][0][1] = 2
a[0][0][2] = 3
a[0][0][3] = 4
a[0][1][0] = 5
a[0][1][1] = 6
a[0][1][2] = 7
a[0][1][3] = 8
a[0][2][0] = 9
a[0][2][1] = 10
a[0][2][2] = 11
a[0][2][3] = 12
a[1][0][0] = 13
a[1][0][1] = 14
a[1][0][2] = 15
a[1][0][3] = 16
a[1][1][0] = 17
a[1][1][1] = 18
a[1][1][2] = 19
a[1][1][3] = 20
a[1][2][0] = 21
a[1][2][1] = 22
a[1][2][2] = 23
a[1][2][3] = 24
```

$$m * N * K + n * K + k$$

$$(m * N + n) * K + k$$

统一形式general：

$a[N_1][N_2][\cdots][N_d]$

$$n_1 * N_2 * \cdots N_{d-1} * N_d +$$
$$n_2 * N_3 * \cdots N_{d-1} * N_d +$$
$$\vdots$$
$$n_k * N_{k+1} * \cdots N_{d-1} * N_d +$$
$$\vdots$$
$$n_{d-1} N_d +$$
$$n_d$$

$$\left(\cdots\left((n_1 * N_2 + n_2) * N_3 + n_3\right) * N_4 \cdots\right) * N_d + n_d$$

5. 高维数组（高阶张量）：存储机制

例6：用下列四维数组（四阶张量）验证数组存储的统一形式。 a[N_1][N_2][…][N_d]

$$(\cdots((n_1 * N_2 + n_2) * N_3 + n_3) * N_4 \cdots) * N_d + n_d$$

$$\begin{aligned} & n_1 * N_2 * \cdots N_{d-1} * N_d + \\ & n_2 * N_3 * \cdots N_{d-1} * N_d + \\ & \vdots \\ & n_k * N_{k+1} * \cdots N_{d-1} * N_d + \\ & \vdots \\ & n_{d-1} N_d + \\ & n_d \end{aligned}$$

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    double cube[2][3][4][5] =
    {
      {   { {1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {11, 12, 13, 14, 15} , {16, 17, 18, 19, 20}},
          { {21, 22, 23, 24, 25}, {26, 27, 28, 29, 30}, {31, 32, 33, 34, 35}, {36, 37, 38, 39, 40}},
          { {41, 42, 43, 44, 45}, {46, 47, 48, 49, 50}, {51, 52, 53, 54, 55}, {56, 57, 58, 59, 60}}
      },
      {   { {61, 62, 63, 64, 65}, {66, 67, 68, 69, 70}, {71, 72, 73, 74, 75} , {76, 77, 78, 79, 80}},
          { {81, 82, 83, 84, 85}, {86, 87, 88, 89, 90}, {91, 92, 93, 94, 95}, {96, 97, 98, 99, 100}},
          { {101, 102, 103, 104, 105}, {106, 107, 108, 109, 110}, {111, 112, 113, 114, 115}, {116, 117, 118, 119, 120}}
      },
    };
    double *p, N1 = 2, N2 = 3, N3 = 4, N4 =5;
    p = cube;
    for(int i =0; i<N1*N2*N3*N4; i++) printf("%d ",*(p+i));
    printf("\n");
    for(int n1=0; n1<N1; n1++)
        for(int n2=0; n2<N2; n2++)
          for(int n3=0; n3<N3; n3++)
            for(int n4=0; n4<N4; n4++)
                printf("a[%d][%d][%d][%d] = %d \n", n1, n2, n3, n4, cube[n1][n2][n3][n4]);
}
```

5. 高维数组（高阶张量）的运算：交换指标

例7：编写一个函数来交换张量A_{ijkl}的j和k两个指标得到B_{ikjl}，以上例的cube[2][3][4][5]作为A_{ijkl}，求B_{ikjl}。

```
void permute(int order, int *rank, double*A, double*B, int j, int k)
{
  int *ranknew; ranknew = (int *) calloc(order,sizeof(int));
  for (int j=0; j<order; j++) ranknew[j] =  rank[j];
  int t = ranknew[j]; ranknew[j] = ranknew[k]; ranknew[k] = t;
  for(int n1=0; n1<rank[0]; n1++)
    for(int n2=0; n2<rank[1]; n2++)
      for(int n3=0; n3<rank[2]; n3++)
        for(int n4=0; n4<rank[3]; n4++)
          { int ns[4]  = {n1, n2, n3, n4};
            int s = ns[j]; ns[j] = ns[k]; ns[k] = s;
            B[((ns[0]*ranknew[1]+ns[1])*ranknew[2]+ns[2])*ranknew[3]+ns[3]]
             = A[((n1*rank[1]+n2)*rank[2]+n3)*rank[3]+n4];
          }
  free(ranknew);
}
```

rank[4] = {2,3,4,5}

弄清楚交换指标的机制！

5. 高维数组（高阶张量）的运算：缩并

例8：编写一个函数来求三阶张量和二阶张量的指标缩并$\sum_j A_{ijk}B_{jl} = C_{ikl}$

```
void contract(int*rankA, int*rankB, int*rankC, double*A, double*B, double*C)
{
   for(int i=0; i<rankC[0]; i++)
      for(int k=0; k<rankC[1]; k++)
          for(int l=0; l<rankC[2];l++)
          {
            double sum = 0;
            for(int j=0; j<rankA[1]; j++)
                sum += A[(i*rankA[1]+j)*rankA[2]+k]*B[j*rankB[1]+l];
           C[(i*rankC[1]+k)*rankC[2]+l]=sum;
          }
}
```

```
double A[2][3][4] =
  {
     { {1, 2, 3, 4}, {5, 6, 7, 8},
       {9, 10, 11, 12}  },
     { {13, 14, 15, 16},
       {17, 18, 19, 20},
       {21, 22, 23, 24} }
  };
double B[3][2] =
        { {1, 2}, {1, 2}, {1, 2} };
double C[2][4][2] =
{
     { {15, 30}, {}, {}, {} },
     { {}, {}, {}, {} }
};
```

请同学先在黑板手算几项，然后与程序对比！

本次课的8道例题作为周五的上机练习！