§12. 指针进阶

1. 基本概念(复习)
2. 变量与指针(复习)
3. 一维数组与指针(复习)
4. 字符串与指针(复习)
5. 返回指针值的函数(复习)
6. 空指针NULL(复习)

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

★ 一维数组的理解方法(下标法、指针法)

一维数组：

```
int a[12]={1,2,3,4,5,6,7,8,9,10,11,12};
```

a ：数组名/数组的首元素地址(⇔&a[0])

由等价关系 a[i] ⇔ *(a+i)可得

&a[i] ：第i个元素的地址 (下标法)

a+i ：第i个元素的地址 (指针法)

a[i] ：第i个元素的值 (下标法)

*(a+i) ：第i个元素的值 (指针法)

&a[i] ⇔ a+i 地址

a[i] ⇔ *(a+i) 值

第0个元素的特殊表示：

a[0] ⇔ *(a+0) ⇔ *a

&a[0] ⇔ a+0 ⇔ a

a	2000	1	a[0]
	2004	2	a[1]
	2008	3	a[2]
	2012	4	a[3]
	2016	5	a[4]
	2020	6	a[5]
	2024	7	a[6]
	2028	8	a[7]
	2032	9	a[8]
	2036	10	a[9]
	2040	11	a[10]
	2044	12	a[11]

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

二维数组：

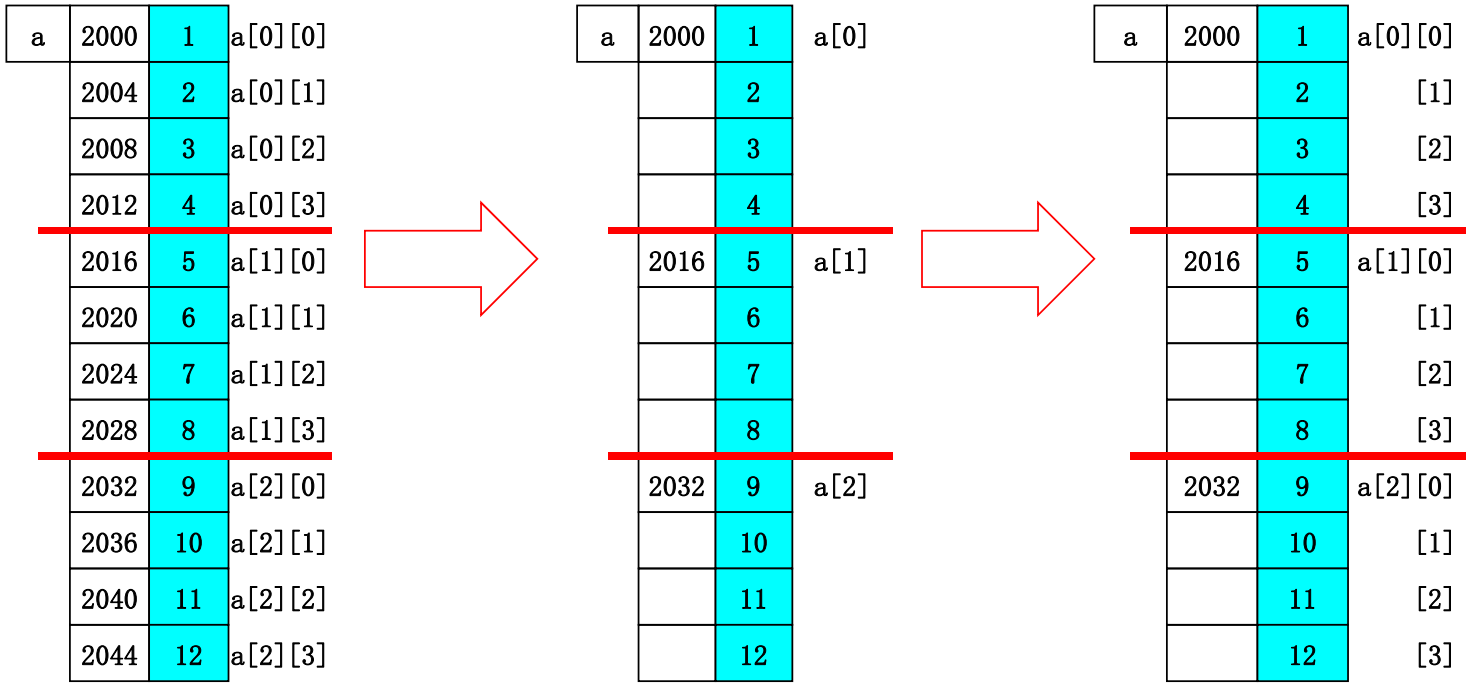
```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
    1   2   3   4
    5   6   7   8
    9   10  11  12
```

第5章的内容：
二维数组 int a[3][4]，
理解为一维数组，有3(行)个元素，
每个元素又是一维数组，有4(列)个元素

a是二维数组名，
a[0],a[1],a[2]是一维数组名

理解1：a | [3][4]

理解2：a[3] | [4]



§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

★ 二维数组加一个下标的理解方法(下标法、指针法)

元素是指
4元素一维数组

```
int a[3][4]={1,…,12};
```

3种理解方法

a	① 二维数组的数组名，即a ② 3元素一维数组的数组名，即a ③ 3元素一维数组的首元素地址,即&a[0]	行地址
&a[i]	3元素一维数组的第i个元素的地址	
a+i	同上	
a[i]	3元素一维数组的第i个元素的值 (即4元素一维数组的数组名 4元素一维数组的首元素的地址)	元素地址
*(a+i)	同上	

i:0-2(行)

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

★ 二维数组加两个下标的理解方法(下标法、指针法)

从第五章概念可知：
a[i][j]：第i行j列元素的值
&a[i][j]：第i行j列元素的地址

令x表示a[i]，则：
x[j]：第i行j列元素的值
&x[j]：第i行j列元素的地址

由一维数组的等价变换可得：
x[j]：第i行j列元素的值
&x[j]：第i行j列元素的地址
*(x+j)：第i行j列元素的值
x+j ：第i行j列元素的地址

所以，用a[i]替换回x，则可得：
a[i][j]:第i行j列元素的值
&a[i][j]:第i行j列元素的地址
*(a[i]+j):第i行j列元素的值
a[i]+j :第i行j列元素的地址

因为：对一维数组 a[i]⇔*(a+i)
所以：*(a[i]+j) ⇔ *(*(a+i)+j)(值)
a[i]+j ⇔ *(a+i)+j (元素地址)

a[i][j]:第i行j列元素的值
&a[i][j]:第i行j列元素的地址
*(a[i]+j):第i行j列元素的值
a[i]+j :第i行j列元素的地址
((a+i)+j):第i行j列元素的值
*(a+i)+j :第i行j列元素的地址

二维数组元素的值和元素的地址均有三种形式：
a[i][j] ⇔ *(a[i]+j) ⇔ *(*(a+i)+j) 值
&a[i][j] ⇔ a[i]+j ⇔ *(a+i)+j 元素地址

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

地址增量的变化规律

对一维数组a：

a+i 实际 a+i*sizeof(基类型)

a	2000	1	a[0]
	2004	2	a[1]
	2008	3	a[2]
	2012	4	a[3]
	2016	5	a[4]
	2020	6	a[5]
	2024	7	a[6]
	2028	8	a[7]
	2032	9	a[8]
	2036	10	a[9]
	2040	11	a[10]
	2044	12	a[11]

对二维数组a[m][n]：

a+i 实际 a+i*n*sizeof(基类型)

a[i]+j 实际 a+(i*n+j)*sizeof(基)

例：a+1：2016　行地址

a[1]+2：2024　元素地址

a	2000	1	a[0][0]
	2004	2	[1]
	2008	3	[2]
	2012	4	[3]
	2016	5	a[1][0]
	2020	6	[1]
	2024	7	[2]
	2028	8	[3]
	2032	9	a[2][0]
	2036	10	[1]
	2040	11	[2]
	2044	12	[3]

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

a	：地址(二维数组/第0行)	
&a[i]	：地址(第i行)	行地址
a+i	：地址(第i行)	
a[i]	：地址(第i行0列)	
*(a+i)	：地址(第i行0列)	
&a[i][j]	：地址(第i行j列)	元素地址
a[i]+j	：地址(第i行j列)	
*(a+i)+j	：地址(第i行j列)	
a[i][j]	：值(第i行j列)	
*(a[i]+j)	：值(第i行j列)	值
((a+i)+j)	：值(第i行j列)	

假设 int a[3][4] 存放在2000开始的48个字节中

2000
2016
2016
2016
2016
2024
2024
2024

假设 i=1
j=2

a+1是地址2016，*(a+1)取a+1的值，还是地址2016
a+1是行地址,*(a+1)取a+1的值,是元素地址

a[2]是地址2032，&a[2]取a[2]的地址，还是2032
a[2]是元素地址，&a[2]取a[2]的地址，是行地址

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

a	：地址(二维数组/第0行)	
&a[i]	：地址(第i行)	行地址
a+i	：地址(第i行)	
a[i]+0	：地址(第i行0列)	
*(a+i)+0	：地址(第i行0列)	元素地址
&a[i][j]	：地址(第i行j列)	
a[i]+j	：地址(第i行j列)	
*(a+i)+j	：地址(第i行j列)	
a[i][j]	：值(第i行j列)	值
*(a[i]+j)	：值(第i行j列)	
((a+i)+j)	：值(第i行j列)	

这两种情况虽然只看到一个下标，但要当做两个下标理解(i行0列的特殊表示)

&a[i]	：地址(第i行)
a+i	：地址(第i行)
a[i]	：地址(第i行0列)
*(a+i)	：地址(第i行0列)

由：&a[i]：行地址　　a[i]：元素地址

　　a+i　：行地址　　*(a+i)：元素地址

得：*行地址　　=> 元素地址（该行首元素）

如何证明?

&首元素地址 => 行地址　（必须首元素!!!）

如何证明?

进一步思考：

(1) &行地址　是什么？ &&行地址呢？

(2) *元素地址 是什么？ **元素地址呢？

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

```
#include <iostream>
using namespace std;
int main()
{   int a[3][4];
    cout <<    a     << endl;   地址a
    cout <<  (a+1)   << endl;   地址a+16
    cout <<  (a+1)+1 << endl;   地址a+32
    cout << *(a+1)   << endl;   地址a+16
    cout << *(a+1)+1 << endl;   地址a+20
    cout <<  a[2]    << endl;   地址a+32
    cout <<  a[2]+1  << endl;   地址a+36
    cout << &a[2]    << endl;   地址a+32
    cout << &a[2]+1  << endl;   地址a+48(已超范围)
    return 0;
}
```

行地址

元素地址

行地址

实际运行一次，观察结果并思考!!!

说明：
每组打印地址后，
再打印地址+1，
目的是区分行地址及元素地址

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

```
#include <iostream>
using namespace std;
int main()
{   int a[3][4];
    cout << sizeof(a)          << endl;
    cout << sizeof(a+1)        << endl;
    cout << sizeof(*(a+1))     << endl;
    cout << sizeof(*(a+1))     << endl;
    cout << sizeof(**(a+1))    << endl;
    cout << sizeof(a[2])       << endl;
    cout << sizeof(*(a[2]))    << endl;
    cout << sizeof(&a[2])      << endl;
    cout << sizeof(*(&a[2]))   << endl;
    return 0;
}
```

另一种验证方法!!!

48		数组大小
4	即&a[1],是地址(指针)	a+1大小
16	指针基类型是int[4]	a+1基类型
16	即a[1],是数组(4元素)	*(a+1)大小
4	数组元素是int	*(a+1)基类型
16	a[2]是数组(4元素)	a[2]大小
4	数组元素是int	a[2]基类型
4	数组a[2]的地址(指针)	&a[2]大小
16	指针基类型是int[4]	&a[2]基类型
	*&a[2]⇔a[2],是数组(4元素)	

行地址 / 元素地址 / 行地址

同

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

7.2. 指向二维数组元素的指针变量

```
#include <iostream>
using namespace std;
int main()
{   int a[3][4], *p;
    p=a[0];
    p=&a[0][0];
    p=*a;
    p=a;
    p=&a[0];
}
```

编译正确，p指向a[0][0]

编译错误，因为a/&a[0]代表的是行地址

```
#include <iostream>
using namespace std;

int main()
{
   int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
   int *p = a[0];
   cout << sizeof(a)  << endl;   48          数组大小
   cout << sizeof(p)  << endl;   4           因为指针
   cout << sizeof(*p) << endl;   4           因为int
   cout << p      << endl;       地址a        元素[0][0]地址
   cout << p+5    << endl;       地址a+20     元素[1][1]地址
   cout << *(p+5) << endl;       6           a[1][1]的值
}
```

a	2000	1	a[0][0]
	2004	2	a[0][1]
	2008	3	a[0][2]
	2012	4	a[0][3]
	2016	5	a[1][0]
	2020	6	a[1][1]
	2024	7	a[1][2]
	2028	8	a[1][3]
	2032	9	a[2][0]
	2036	10	a[2][1]
	2040	11	a[2][2]
	2044	12	a[2][3]

假设a的首地址是2000，则区别如下：

p=a[0]：p的值是2000，基类型是int，p+1的值为2004

p=a　：p的值是2000，基类型是int*4，p+1的值为2016

因为p是基类型为int的指针变量，所以：

p+i ⇔ p+i*sizeof(int)

p+5 ⇔ &a[1][1]

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.1. 二维数组的地址

7.2. 指向二维数组元素的指针变量

例：打印二维数组的值（以下四种方法均正确，均是按一维方式顺序循环）

```
int main()
{  int a[3][4]={...}, *p;
   for(p=a[0];p<a[0]+12;p++)
      cout << *p << ' ';
   cout << endl;
   return 0;
}
```

```
int main()
{  int a[3][4]={...};
   int i, j, *p = a[0];
   for(i=0; i<3; i++)
     for(j=0; j<4;j++)
        cout << *p++ << ' ';
   return 0;
}
```

```
int main()
{  int a[3][4]={...};
   int i, j, *p=&a[0][0];
   for(i=0; i<12; i++)
      cout << *p++ << ' ';
   return 0;
}
```

```
int main()
{  int a[3][4]={...}
   int i, j, *p=&a[0][0];
   for(; p-a[0]<12;)
     cout << *p++ << ' ';
   return 0;
}
```

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.3. 指向由m个元素组成的一维数组的指针变量

```
#include <iostream>
using namespace std;
int main()
{   int a[3][4], (*p)[4];
    p=a[0];
    p=&a[0][0];
    p=*a;
    p=a;
    p=&a[0];
}
```

编译错误

编译正确

int a[3][4]={…};

int (*p)[4]=a;

(*p)有4个元素

每个元素类型是int

=> p是指向4个元素组成的一维数组的指针

int a[4]

a有4个元素

每个元素类型是int

*p+j / *(p+0)+j:取这个一维数组中的第j个元素

p+i 实际 p+i*4*sizeof(int)

*(p+i)+j 实际 p+(i*4+j)*sizeof(int)

★ 使用：

p：地址(m个元素组成的一维数组的地址)

*p：值(是一维数组的名称，即一维数组的首元素地址)

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.3. 指向由m个元素组成的一维数组的指针变量

int a[3][4]={1,…,12}, (*p)[4] ;

p = a;

p+1 : 行地址2016(a[1])

*p+1 : 元素地址2004(a[0][1])

*(*p+1) : 元素值2(a[0][1])

*(p+1)+2 : 元素地址2024(a[1][2])

((p+1)+2): 元素值7(a[1][2])

p是行地址2000

*p是元素地址2000

p+1是行地址2016

*(p+1)是元素地址2016

a	2000	1	a[0][0]
	2004	2	a[0][1]
	2008	3	a[0][2]
	2012	4	a[0][3]
	2016	5	a[1][0]
	2020	6	a[1][1]
	2024	7	a[1][2]
	2028	8	a[1][3]
	2032	9	a[2][0]
	2036	10	a[2][1]
	2040	11	a[2][2]
	2044	12	a[2][3]

```
#include <iostream>
using namespace std;
int main()
{  int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
   int (*p)[4] = a;
   cout << sizeof(a)  << endl;
   cout << sizeof(p)  << endl;
   cout << sizeof(*p) << endl;
   cout << p       << endl;
   cout << p+1     << endl;
   cout << *p      << endl;
   cout << *p+1    << endl;
   cout << *(*p+1) << endl;
}
```

```
48          数组大小
4           因为指针
16          因为int[4]
地址a       行地址
地址a+16    +1 = +16
地址a       元素地址
地址a+4     +1 = +4
2           a[0][1]的值
```

```
#include <iostream>
using namespace std;

int main()
{
    int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
    int (*pl)[4], *p;
    for (pl=a; pl < a+3; pl++) {     //行指针
        for (p=*pl; p < *pl+4; p++) //元素指针
            cout << *p << ' ';
        cout << endl; //每行一个回车
        }
}
```

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.4. 用指向二维数组元素的指针做函数参数

★ 形参是对应类型的简单指针变量

```
#include <iostream>
using namespace std;

void fun(int *data)
{
    if (*data%2==0)
        cout << *data << endl;
}
int main()
{
    int a[3][4]={...}, *p;
    for(p=a[0]; p<a[0]+12; p++)
        fun(p);
    cout << endl;

    return 0;
}
```

实参是指向二维数组元素的指针变量
形参是对应类型的简单指针变量

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.5. 用指向二维数组的指针做函数参数

思考：若f1/f2/f3中为sizeof(**x1/**x2/**x3)
则：结果是多少？为什么？

5.4. 用数组名作函数参数

5.4.3. 用多维数组名做函数实参

当时第5章的说法，都不准确，形参数组不存在
形参的本质是指针变量

★ 形参为相应类型的多维数组

★ 实、形参数组的列必须相等，形参的行可以不指定，或为任意值(实参传入二维数组的首地址，只要知道每行多少列实形参即可对应，不关心行数)

```
#include <iostream>
using namespace std;
void f1(int x1[][4])     //形参数组不指定行大小
{   cout << "x1_size=" << sizeof(x1) << endl;
}
void f2(int x2[3][4])    //形参数组行大小与实参相同
{   cout << "x2_size=" << sizeof(x2) << endl;
}
void f3(int x3[123][4]) //形参数组行大小与实参不同
{   cout << "x3_size=" << sizeof(x3) << endl;
}
int main()
{   int a[3][4];
    cout << "a_size=" << sizeof(a) << endl;
    f1(a);
    f2(a);
    f3(a);
}
```

```
a_size=48
x1_size=4    因为 *
x2_size=4    因为 *
x3_size=4    因为 *
```

```
#include <iostream>
using namespace std;
void f1(int x1[][4])     //形参数组不指定行大小
{   cout << "x1_size=" << sizeof(*x1) << endl;
}
void f2(int x2[3][4])    //形参数组行大小与实参相同
{   cout << "x2_size=" << sizeof(*x2) << endl;
}
void f3(int x3[123][4]) //形参数组行大小与实参不同
{   cout << "x3_size=" << sizeof(*x3) << endl;
}
int main()
{   int a[3][4];
    cout << "a_size=" << sizeof(a) << endl;
    f1(a);
    f2(a);
    f3(a);
}
```

```
a_size=48
x1_size=16   因为int[4]
x2_size=16   因为int[4]
x3_size=16   因为int[4]
```

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.5. 用指向二维数组的指针做函数参数

★ 形参是指向m个元素组成的一维数组的指针变量

★ 形参是相应类型的二维数组

(行的大小可省略，本质上仍然是指向m个元素组成的一维数组的指针变量)

例：二维数组名做实参

```
void output(int (*p)[4])
{
    int i,j;
    for(i=0; i<3; i++)
       for(j=0; j<4; j++)
          cout << *(*(p+i)+j) << " ";
    cout << endl;
}

int main()
{
    int a[3][4]={...};
    output(a);
    return 0;
}
```

int p[3][4]
int p[][4]
int p[123][4]
本质都是行指针变量

((p+i)+j)
*(p[i]+j)
p[i][j]
二维数组值的三种形式

实参是二维数组名
形参是指向m个元素的一维数组的指针变量

3. 一维数组与指针中

★ 对一维数组而言，数组的指针和数组元素的指针，其实都是指向数组元素的指针变量(特指0/任意i)，因此本质相同(基类型相同)

★ 数组名代表数组首地址，指针是地址，但本质不同(sizeof(数组名)/sizeof(指针)大小不同)

本处：

★ 对二维数组而言，数组的指针是指向一维数组的指针，数组元素的指针是指向单个元素的指针，两者的本质是完全不同的(基类型不同)

§12. 指针进阶

7. 多维数组与指针(补充，极其重要!!!)

7.5. 用指向二维数组的指针做函数参数

★ 形参是指向m个元素组成的一维数组的指针变量

★ 形参是相应类型的二维数组

(行的大小可省略，本质上仍然是指向m个元素组成的一维数组的指针变量)

二维数组做函数参数的实参/形参的四种组合

```
//形参是二维数组名
void fun(int p[][4])
{
   ...
}
```

```
//形参是指向m个元素组成的
  一维数组的指针变量
void fun(int (*p)[4])
{
   ...
}
```

```
//实参是二维数组名
int main()
{   int a[3][4]={...};

    fun(a);
}
```

```
//实参是指向m个元素组成的
  一维数组的指针变量
int main()
{   int a[3][4]={...};
    int (*p)[4];
    p=a;
    fun(p);
}
```