二级指针和二维数组

二维数组

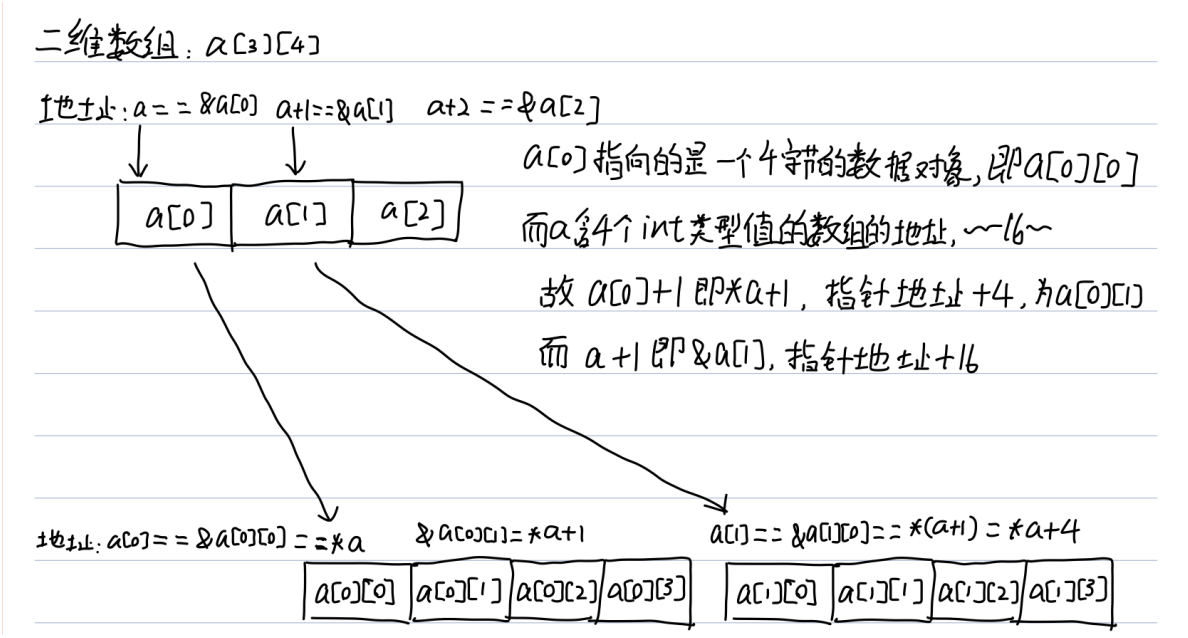
二维数组本质就是一维数组，只不过所存的每个元素是数组(注意是数组而不是指针)，因此一个二维指针a[3][4]的第一个元素是a[0]而不是a[0][0]，故数组名a不是a[0][0]的地址而是a[0]的地址

即：

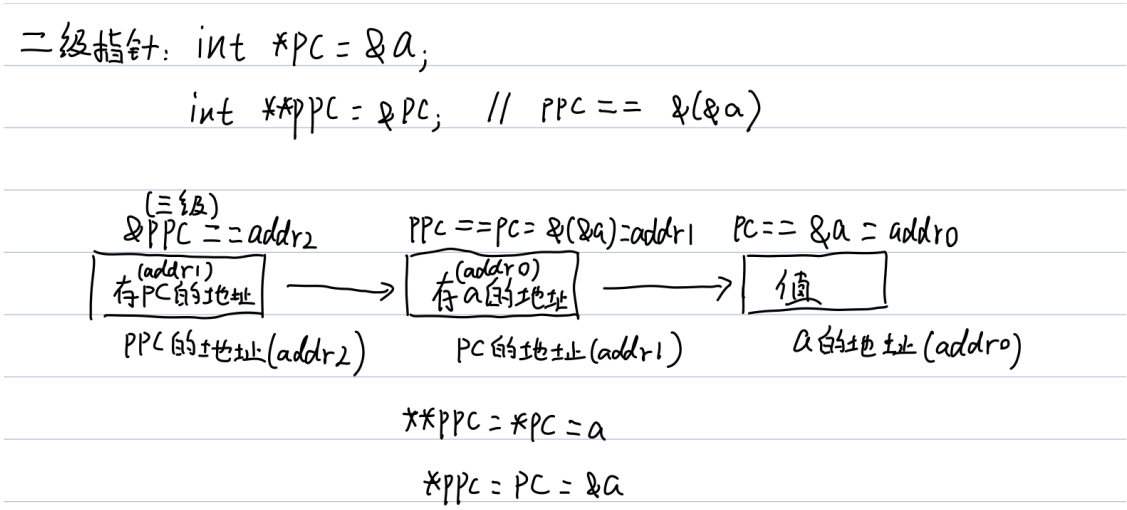
```
a == &a[0];
a[0] == &a[0][0];
a == &(&a[0][0]);  // 数组名a的类型其实为int (*)[N]，有别于二级指针，虽然都是解两次地址获取值

// 定义一个指针变量p
int *p;
p = a;  // 错误，两者数据类型不同
p = a[0];
p = &a[0][0];
*a == *(&(&a[0][0])) == &a[0][0] == p;
**a = a[0][0];

// 注意操作的细节区别
a+i == &a[i];
p+i == &a[0][i];
```



二级指针



详见如下**例子**

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <time.h>

#define K 7  // 原始数据块个数
#define NUM_OF_REDUN 1  // 冗余块个数
#define NUM_OF_BLOCK(K, NUM_OF_REDUN) K+NUM_OF_REDUN  // 数据块总个数
#define M 256
#define L 16  // 循环移位矩阵大小
#define BLOCK_SIZE(M, L) M*L   // 数据块的大小，单位比特


int ofs_encode_P(int n, int m, int blkbytes, uint8_t **data);  // 声明编码函数，n为原始数据块个数，m为冗余块个数，blkbytes为数据块大小，单位为字节，__pptrs指向数据块
void random255(uint8_t (*a)[BLOCK_SIZE(M, L)], int n, int min_num, int max_num);
 // 生成单个随机数据，a为待生成数据数组，n为行数组大小，后两个为上下限


int main()
{


    uint8_t data_block[NUM_OF_BLOCK(K, NUM_OF_REDUN)][BLOCK_SIZE(M, L)] = {0};
 // 单个原始数据块
    printf("addr00 = %a\n", &data_block);
    printf("addr0 = %a\n", data_block);
    printf("addr1 = %a\n", &data_block[0]);
    printf("addr2 = %a\n", &data_block[0][0]);
    uint8_t *row[K];  // 重新申请一个指针数组，指向二维数组的每一行的首地址，以代替形参 int (*pt)[3] 或 int pt[][3]即行指针的作用
    for(int i = 0; i < K; i++)
        row[i] = data_block[i];

    random255(data_block, BLOCK_SIZE(M, L), 0, 255);

    ofs_encode_P(K, NUM_OF_REDUN, BLOCK_SIZE(M, L)/8, row);



    return 0;
```

```
}


int ofs_encode_P(int n, int m, int blkbytes, uint8_t **data)
{
    int i, j = 0;
    int m_len = blkbytes*8;


    printf("addr0 = %a\n", &data);
    printf("addr1 = %a\n", data);
    printf("addr2 = %a\n", *data);
    printf("value = %d\n", **data);

    int t = 1;
    // 生成P包(第一冗余包)
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < m_len; j++)
            *(*(data+n) + j) = *(*(data+n) + j) ^ *(*(data+i) + j);
    }

    // 生成Q包(第二冗余包)
    // 生成R包(第三冗余包)
    return 0;
}

void random255(uint8_t (*a)[BLOCK_SIZE(M, L)], int n, int min_num, int max_num)
{
    int i = 0;
    int j = 0;
    srand(time(0));  //设置时间种子
    for(i = 0; i < K; i++)
    {
        for(j = 0; j < n; j++)
            *(*(a+i) + j) = (uint8_t)(rand() % (max_num - min_num +1 ) +
min_num);
    }
}
```

```
addr00 = 0xa.a1f7ef2ap-1039
addr0 = 0xa.a1f7ef2ap-1039
addr1 = 0xa.a1f7ef2ap-1039
addr2 = 0xa.a1f7ef2ap-1039
addr0 = 0xa.a1f7ef21p-1039
addr1 = 0xa.a1f7ef22p-1039
addr2 = 0xa.a1f7ef2ap-1039
value = 4
```

```
/*
&data_block == data_block == &data_block[0] == &data[0][0]
([0][0]位置比较特殊，两层地址都相等)
data的值和data_block的值不一致，说明两者指向不同的地址
*data == data_block，仅从[0][0]处还看不出问题
有一个有意思的现象，对二维数组数组名data_block做取址运算，得到的结果和data_block本身相等，但是对二级指针data做取址运算，得到的结果和data本身不相等
*/
```

例子2

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

#define K 7  // 原始数据块个数
#define NUM_OF_REDUN 1  // 冗余块个数
#define NUM_OF_BLOCK(K, NUM_OF_REDUN) K+NUM_OF_REDUN  // 数据块总个数
#define M 256
#define L 16  // 循环移位矩阵大小
#define BLOCK_SIZE(M, L) M*L   // 数据块的大小，单位比特


int ofs_encode_P(int n, int m, int blkbytes, uint8_t **data);  // 声明编码函数，n为原始数据块个数，m为冗余块个数，blkbytes为数据块大小，单位为字节，__pptrs指向数据块
void random255(uint8_t (*a)[BLOCK_SIZE(M, L)], int n, int min_num, int max_num);
 // 生成单个随机数据，a为待生成数据数组，n为行数组大小，后两个为上下限


int main()
{


    uint8_t data_block[NUM_OF_BLOCK(K, NUM_OF_REDUN)][BLOCK_SIZE(M, L)] = {0};
 // 单个原始数据块
    printf("addr00 = %a\n", &data_block+1);
    printf("addr0 = %a\n", data_block+1);
    printf("addr1 = %a\n", &data_block[1]);
    printf("addr2 = %a\n", &data_block[1][2]);
    uint8_t *row[K];  // 重新申请一个指针数组，指向二维数组的每一行的首地址，以代替形参 int (*pt)[3] 或 int pt[][3]即行指针的作用
    for(int i = 0; i < K; i++)
        row[i] = data_block[i];

    random255(data_block, BLOCK_SIZE(M, L), 0, 255);

    ofs_encode_P(K, NUM_OF_REDUN, BLOCK_SIZE(M, L)/8, row);



    return 0;
}


int ofs_encode_P(int n, int m, int blkbytes, uint8_t **data)
```

```
{
    int i, j = 0;
    int m_len = blkbytes*8;


    printf("addr1 = %a\n", data+1);
    printf("addr2 = %a\n", *(data+1));
    printf("value = %d\n", *(*(data+1)+2));
    int t = 1;
    // 生成P包(第一冗余包)
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < m_len; j++)
            *(*(data+n) + j) = *(*(data+n) + j) ^ *(*(data+i) + j);
    }

    // 生成Q包(第二冗余包)
    // 生成R包(第三冗余包)
    return 0;
}

void random255(uint8_t (*a)[BLOCK_SIZE(M, L)], int n, int min_num, int max_num)
{
    int i = 0;
    int j = 0;
    srand(time(0));  //设置时间种子
    for(i = 0; i < K; i++)
    {
        for(j = 0; j < n; j++)
            *(*(a+i) + j) = (uint8_t)(rand() % (max_num - min_num +1 ) +
min_num);
    }
}
```

```
printf("addr00 = %a\n", &(data_block+1));
报错：lvalue required as unary '&' operand
    目前理解：还在寄存器中，要先MOV到内存里才能取值
```

```
addr00 = 0xb.75c3ffbep-1038
addr0 = 0xb.75c3f8bep-1038
addr1 = 0xb.75c3f8bep-1038
addr2 = 0xb.75c3f8be2p-1038
addr1 = 0xb.75c3f7ba8p-1038
addr2 = 0xb.75c3f8bep-1038
value = 219
```

```
/*
&data_block+1 != data_block+1
data_block+1 == &data_block[1]
data+1 != data_block+1，跟上述结论一样，两者指向不同地址
*(data+1) == &data_block[1]的值一致，说明两者指向同一个地址
*(*(data+1)+2) == data_block[1][2]，说明*(data+1)+2指到了data_block[1][2]的地址

有意思的现象2，data+1和data_block+1虽然不指向同一个地址，但是*(data+1)和data_block+1指向了同一个地址，理论是应该*(data+1)和*(data_block+1)指向同一个地址
*/
```

关于现象2的解释**（解释不了）**：

如下`data_block+1 == data_block[1] == &data_block[1] == &data_block[1][0]`

理论上`data_block+1 == &data_block[1]`和`data_block[1] == &data_block[1][0]`这两个是正常的，但是`data_block[1]`和**其地址相等有问题了**

`data_block+1 == &data_block[1]`，即`data_block+1 指向 data_block[1]`，而`data_block[1] == *(data_block+1)`

```
printf("addr0 = %a\n", data_block+1);
printf("addr1 = %a\n", data_block[1]);
printf("addr1_s = %a\n", &data_block[1]);
printf("addr1_sd = %a\n", &data_block[1][0]);
```

```
addr0 = 0xe.9ef3f89cp-1038
addr1 = 0xe.9ef3f89cp-1038
addr1_s = 0xe.9ef3f89cp-1038
addr1_sd = 0xe.9ef3f89cp-1038
```

两者关系（其实没有什么关系，只是操作很类似导致的误区）

虽然看着很相似，但是有区别

两者作为函数参数时，并不等价。在二级指针作为函数形参时，二维数组名不能当参数传入

产生这样误解的原因：一维数组作为函数参数时，编译器会把数组名当作指向该数组的一个元素的指针，但在二维数组上并不适用

因此需要一个**数组指针(指向数组的指针)**作为函数形参来接收二维数组传入

pt相当于行指针，等价于二维数组的

```
void somefunction(int (*pt)[4]);  // 即pt是一个指向数组的指针，指向内含4个int类型数据的数组
// 或者
voide somefunction(int pt[][4]);

pt+i == &a[i];  // 相当于a+i
```

- 如果需要**二级指针**作形参，能作为函数实参的是**二级指针**（例如：类型为`int **`），**指针数组**（例如：类型为`int *[]`），**一级指针的地址**

如果需要传入二维数组那么可以通过如下方式处理：

指针数组(数组里存指针)做如下处理

```
void somefunction(int **pt);

int data_block[K][2];
int *row[K];  // 重新申请一个指针数组，指向二维数组的每一行的首地址，以代替形参 int (*pt)[3] 或 int pt[][3]即行指针的作用

// 即将二维数组每行的首地址存在指针数组里
for(int i = 0; i < K; i++)
        row[i] = data_block[i];

somefunction(row);
```

- 当**数组指针**作为函数形参时，能作为函数实参的是**二维数组**（例如：类型为`int(*)[N]`），**数组指针**（例如：类型为`int(*)[N]`）
- 当**二维数组**作为函数形参时，能作为函数实参的是二维数组，**数组指针**
- 当指针数组作为函数形参时，能作为函数实参的是**指针数组**，**二级指针**，**一级指针的地址**

void型指针

void *

`void *`是一种指针类型，常用在`函数参数、函数返回值`中需要兼容不同指针类型的地方。我们可以将别的类型的指针**无需强制类型转换**的赋值给`void *`类型。也可以将`void *`强制类型转换成任何别的指针类型，至于强转的类型是否合理，就需要我们程序员自己控制了。

```
#include<stdio.h>

int main(int argc,char *argv[])
{
    int a = 2;
    double b = 2.0;
    void *c;                    //定义void *
    int *p = &a;
    c = p;                      //将int * 转成void *,
    double *q = (double *)c;  //将void *转成double *
    printf("%.f\n",*q);
    printf("a:%p b:%p q:%p\n", &a, &b, q);

    return 0;
}
```

关于上述例子，其实我本身想表达指针类型确定它解释的数据范围这个观点，也就是`int *`解释的范围是4个字节，但是`double *`解释的范围是8个字节，所以上述例子中，`int *`转了`void *`再到`double *`，从而将它解释的范围扩大到了8个字节，那应该输出什么结果呢？答案是从 a 的地址开始，下面 8 个字节组成的`double`的数值，a的地址解释为`int`是2，但是再加上4个字节，这4个字节的内存值是不确定，脏数据，因此最后的输出也是脏数据。

void **

不同于void*

传参时，同上的**二级指针要求**，但是需要先转换为void类型，在函数内使用是，再转换为相应的数据类型

第一个例子

```
    int i1 = 1;
    //int* pi1 = &i1;
    //int** ppi1 = &pi1;
    void* pv1 = &i1;
    void** ppv1 = &pv1;
    int** ppout;
    cout << i1 <<endl;
    cout << &i1 <<endl;
    cout << ppv1 <<endl;
//  cout << *ppv1 << endl;  //非法，void*型指针不能直接解引用
//  cout << **ppv1 << endl; //非法，void*型指针不能直接解引用
    pv1 = (int*)pv1;
//  ppv1 = (int**)ppv1;     //非法，cannot convert from 'int ** ' to 'void ** '
    ppout = (int**)ppv1;
    cout << *ppv1 << endl;
    cout << **ppout << endl;
```

第二个例子

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

#define K 2  // 原始数据块个数
#define N 10

void tt(void **__pptrs);
void random255(int (*a)[N], int n, int min_num, int max_num);  // 生成单个随机数据，a为待生成数据数组，n为行数组大小，后两个为上下限

int main()
{
    int data_block[K][N] = {0};  // 单个原始数据块

    void *row[K];  // 重新申请一个指针数组，指向二维数组的每一行的首地址，以代替形参 int (*pt)[3] 或 int pt[][3]即行指针的作用
    int min = 1;
    int max = 10;

    for(int i = 0; i < K; i++)
        row[i] = data_block[i];

    random255(data_block, N, min, max);
    tt(row);
    return 0;
}
```

```c
void tt(void **__pptrs)
{
    int a = 0;
    int **__pptrs_tmp = (int**)__pptrs;  // 强制类型转换
    int t = 1;
    a = *(*(__pptrs_tmp+1) + 1);
    printf("a = %d", a);
}

void random255(int (*a)[N], int n, int min_num, int max_num)
{
    int i = 0;
    int j = 0;
    srand(time(0));  // 设置时间种子
    for(i = 0; i < K; i++)
    {
        for(j = 0; j < N; j++)
            *(*(a+i) + j) = rand() % (max_num - min_num + 1) + min_num;
    }
}
```