In C Language How to Use Pointer to Access Two-Dimensional Array Element

WeiQing Bai
College of Software and Technology
QingDao University
QingDao, China
bwqsy1@sina.com

Abstract—In C language, it is difficult to use the pointer to access the two-dimensional array element. The reason is the addresses of two-dimensional array are many, pointers that can access the two-dimensional array element are many and complex. This paper analyzes the two-dimensional array address and pointer, introduces various forms of using pointer to express two-dimensional array element, explains the relationship between pointer and two-dimensional array address. The program examples show how to use pointers to access the two-dimensional array element.

Keywords—C language; two-dimensional array; address; pointer

I. INTRODUCTION

In C, to learn pointer is difficult, students use pointer to access two-dimensional array element feel especially difficult. The reason is the addresses of two-dimensional array are many, pointers that can access the two-dimensional array element are many and complex. In order to solve this problem, this paper first analyzes the address of one-dimensional array and how to use pointer to express one dimensional array element. Based on this, it further analyzes the two-dimensional array address and pointer, introduces various forms of using pointer to express two-dimensional array element, explains the relationship between the pointer and two-dimensional array address.

II. ONE-DIMENSIONAL ARRAY AND POINTER

A one-dimensional array and a pointer which points to one-dimensional array are defined as follows:[1]

```c
int a[10],*p;
```

A. the Address of One-dimensional Array

One-dimensional array involves the following address:

1. the address of array element:&a[i] (0≤i≤9)
2. array name a, the name of an array is a synonym for the location of the initial element:a = &a[0].

B. the Pointer Variables Related With the One-dimensional Array

If p is a pointer to an integer, defined as: int *p;
then the assignment p = &a[0];
sets p to point to element zero of a, we can access the array element through pointer p.

C. Various Forms of One-dimensional Array Element Address

We can use subscript or pointer to express one-dimensional array element address, there has the following four kind of representation:

1. subscript method: a[i]
2. array name method: a+i
3. pointer method: p+i
4. pointer subscript method: &p[i]

D. Various Forms of One-dimensional Array Element

We can use subscript or pointer to express one-dimensional array element, there has the following four kind of representation:

1. subscript method: a[i]
2. array name method: *(a+i)
3. pointer method: *(p+i)
4. pointer subscript method: *p[i]

One-dimensional array is the basis of learning two-dimensional array, if we understand the relationship between pointer and one-dimensional array, then we can easily understand the relationship between pointer and two-dimensional array.

III. TWO-DIMENSIONAL ARRAY AND POINTER

A two-dimensional array is defined as follows:[1]

```c
int a[3][4];
```

A. the Address of Two-dimensional Array

In C, a two-dimensional array is really a one-dimensional array, each of whose elements is a one-dimensional array. For example, two-dimensional array a has three elements a[0], a[1], a[2], from the knowledge of one-dimensional array we know that a and &a[0] are equivalent, a+1 and &a[1] are equivalent, a+2 and &a[2] are equivalent.

We can sum up that a+i and &a[i] are equivalent (0≤i≤2), therefore *(a+i) and a[i] are equivalent. A+i and &a[i] are starting address of each row in the array, they are called row address.[2]

Each element of array a is a one-dimensional array, such as a[0][4], a[1][4], a[2][4], wherein a[0], a[1], a[2] can be regarded as the name of the one-
dimensional array. Therefore, \( a[0] \) and \&\( a[0][0] \) are equivalent, \( a[0]+1 \) and \&\( a[0][1] \) are equivalent, \( a[0]+2 \) and \&\( a[0][2] \) are equivalent, \( a[0]+3 \) and \&\( a[0][3] \) are equivalent.

\( a[1] \) and \&\( a[1][0] \) are equivalent, \( a[1]+1 \) and \&\( a[1][1] \) are equivalent, \( a[1]+2 \) and \&\( a[1][2] \) are equivalent, \( a[1]+3 \) and \&\( a[1][3] \) are equivalent.

\( a[2] \) and \&\( a[2][0] \) are equivalent, \( a[2]+1 \) and \&\( a[2][1] \) are equivalent, \( a[2]+2 \) and \&\( a[2][2] \) are equivalent, \( a[2]+3 \) and \&\( a[2][3] \) are equivalent.

We can sum up that \( a[i] \) and \&\( a[i][0] \) are equivalent, \( a[i]+j \) and \&\( a[i][j] \) are equivalent (0 ≤ \( i \) ≤ 2, 0 ≤ \( j \) ≤ 3).

\( *\( a+i \) \), \( a[i] \), \( a[i]+j \), \( \&\( a[i][j] \) \) are array element address, they are called element address.

Through the analysis that \( a \) and \&\( a[0] \) and \*\( a \) and \( a[0] \) and \&\( a[0][0] \) have identical values, but they mean different meaning. \( a \) and \&\( a[0] \) are the address of row zero of the array, \( a[0] \) and \&\( a[0][0] \) are the address of element of column zero of row zero.

\( a+i \) and \&\( a[i] \) and \*\( (a+i) \) and \( a[i] \) and \&\( a[i][0] \) have identical values, but they mean different meaning. \( a+i \) and \&\( a[i] \) are the address of row \( i \) of the array, \( a[i] \) and \&\( a[i][0] \) are the address of element of column zero of row \( i \).

Those who need a specification is that \( a[i] \) is not an actual variable, \&\( a[i] \) is just a calculated address, is the address of row \( i \). \( *(a+i) \) is not the content of \( a+i \), is a form of the address.

In summary, the address of two-dimensional array is divided into row address and element address two categories.

1) Row Address

There are two forms of row address:

\( a+i \)  
\&\( a[i] \)

The name of two-dimensional array is the address of row zero, \( a=a[0] \).

2) Element Address

There are several forms of element address:[3]

\( \begin{align*}  
1) &a[i][j] & 2) *(a+i)+j \\
3) &a[i][0]+j & 4) &a[i][j] \\
5) &*(a+i)[j] & 6) &\&a[i][0][i] \\
7) &a[0][4*i+j] & 8) &a[0][0]+4*i+j \\
9) &a+4*i+j \\
\end{align*} \)

Among them, \( 1) \sim 6 \) are the form of two-dimensional, \( 7) \sim 9 \) are the form of one-dimensional.

B. the Pointer Variables Related With the Two-dimensional Array

The pointer variables related with the two-dimensional array are many, will be introduced below.

1) the Pointer Variables Which Point to the Two-dimensional Array Element

The pointer variable which points to the two-dimensional array element is defined as:

\[
\text{int a[3][4],*p;}
\]

a) Assignment

Pointer \( p \) points to the element of column zero of row zero of the two-dimensional array, the correct assignment is:

\[
p=a[0]; \
\]

the wrong assignment is:

\[
p=a; \
\]

Because pointer \( p \) is defined as a pointer variable which points to the two-dimensional array element, so \( p \) only can be assigned a element address, not a row address.

b) Using Pointer \( p \) to Express Element Address of Two-dimensional Array

After the assignment

\[
p=a[0]; \
\]

there are two forms which use pointer \( p \) to express element address of two-dimensional array:

\[
1) \ p+4*i+j \\
2) \ &p[4*i+j]
\]

Among them, \( i \) is row subscript of the array, \( j \) is column subscript of the array. \( i \) and \( j \) that appear below is the same meaning with this.

c) Using Pointer \( p \) to Express Element of Two-dimensional Array

After the assignment

\[
p=a[0]; \
\]

there are two forms which use pointer \( p \) to express element of two-dimensional array:

\[
1) \ *(p+4*i+j) \\
2) \ p[4*i+j]
\]

d) Application

Example 1

\[
\text{main()}
\]

\[
\quad \text{int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12},*p;}
\]

\[
\quad \text{for(p=a[0];p<a[0]+12;p++)}
\]

\[
\quad \text{printf("%d ",*p);}
\]

In this application, pointer \( p \) is used to control cycle index. During the cycle process, the value of \( p \) is changed, pointer \( p \) is moving. Because \( p \) is the pointer variable which points to the two-dimensional array element, so after \( p++ \), \( p \) points to the next element.

Example 2

\[
\text{main()}
\]

\[
\quad \text{int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12},*p,i,j;}
\]

\[
\quad \text{p=&a[0][0];}
\]

\[
\quad \text{for(i=0;i<3;i++)}
\]

\[
\quad \text{for(j=0;j<4;j++}
\]

\[
\quad \text{printf("%d ",*p);}
\]

\[
\quad \text{p++;}
\]

In this application, \( i \) and \( j \) are used to control cycle index. During the cycle process, the value of \( p \) is
changed, pointer p is moving. After p++, p points to the next element.

Example 3

```c
main()
{
    int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12},*p,i,j;
    p=a;
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            printf("%d ",*(p+4*i+j));
}
```

In this application, i and j are used to control cycle index. During the cycle process, the value of p is not changed, pointer p is not moving.

2) The Pointer Variables Which Point to the One-dimensional Array

The pointer variable which points to the one-dimensional array is defined as:

```c
int a[3][4],(*p)[4];
```

It should be noted that the size of the one-dimensional array which pointer p points to should be equal to the number of columns of the two-dimensional array.

a) Assignment

Pointer p points to row zero of the two-dimensional array, the correct assignment is:

```c
p=a; or p=&a[0];
```

the wrong assignment is:

```c
p=a[0]; or p=&a[0][0];
```

Because pointer p is defined as a pointer variable which points to the one-dimensional array, so p only can be assigned a row address, not an element address.

b) Using Pointer p to Express Element Address of Two-dimensional Array

After the assignment

```c
p=a; or p=&a[0];
```

there are several forms which use pointer p to express element address of two-dimensional array:

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

c) Using Pointer p to Express Element of Two-dimensional Array

After the assignment

```c
p=a; or p=&a[0];
```

there are several forms which use pointer p to express element of two-dimensional array:

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

d) Application

Example 4

```c
main()
{
    int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12},(*p)[4],i,j;
    p=&a[0];
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            printf("%d ",*(p+i+j));
}
```

In this application, i and j are used to control cycle index. During the cycle process, the value of p is not changed, pointer p is not moving.

Example 5

```c
main()
{
    int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12},(*p)[4],i,j;
    p=a;
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            printf("%d ",*(p+i+j));
    p++;
}
```

In this application, i and j are used to control cycle index. During the outside loop process, the value of p is changed, pointer p is moving. Because p is the pointer variable which points to the one-dimensional array, so after p++, p points to the next row.

3) Pointer Array

Pointer array is defined as:

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

It should be noted that the size of the pointer array should be equal to the number of rows of the two-dimensional array.

a) Assignment

Pointer of pointer array separately points to the element of column zero of each row of the two-dimensional array, the correct assignment is:

```c
for(i=0;i<3;i++)   p[i]=a[i];  
```

or for(i=0;i<3;i++)   p[i]= *(a+i);  

or for(i=0;i<3;i++)    p[i]= &a[i][0];  

the wrong assignment is:

```c
for(i=0;i<3;i++)    p[i]=&a[i];  
```

or for(i=0;i<3;i++)     p[i]=a+i; 

Because the pointer of pointer array is defined as a pointer variable which points to the element of the two-dimensional array, so pointer array element p[i] only can be assigned an element address, not a row address. It should be noted that p is the pointer array name, not a variable, constructions like p=a is illegal.

b) Using Pointer of Pointer Array to Express Element Address of Two-dimensional Array
After the assignment
for(i=0;i<3;i++)   p[i]=a[i];
or for(i=0;i<3;i++)   p[i]= *(a+i);
or for(i=0;i<3;i++)   p[i]= &a[i][0];

there are several forms which use pointer of pointer array to express element address of two-dimensional array:
①   p[i]+j
②   *(p+i)+j
③   &p[i][0]+j
④   &p[i][j]
⑤   &*(p+i)[j]
⑥   &(&p[i][0])[j]

3) c) Using Pointer of Pointer Array to Express Element of Two-dimensional Array

After the assignment
for(i=0;i<3;i++)   p[i]=a[i];
or for(i=0;i<3;i++)  p[i]= *(a+i);
or for(i=0;i<3;i++)   p[i]= &a[i][0];

there are several forms which use pointer of pointer array to express element of two-dimensional array:
①   *(p[i]+j)
②   *(*(p+i)+j)
③   *(&p[i][0]+j)
④   p[i][j]
⑤   *(p+i)[j]
⑥   (&p[i][0])[j]

4) d) Application

Example 6
main()
{
  int a[3][4]= {1,2,3,4,5,6,7,8,9,10,11,12}, *p[3],**q,i,j;
  for(i=0;i<3;i++)
    p[i]=&a[i][0];
  q=p; or q=&p[0];
  there are several forms which use pointer q to express element address of two-dimensional array:
  ① q[i]+j
  ② *(q+i)+j
  ③ &q[i][0]+j
  ④ &q[i][j]
  ⑤ &(*(q+i))[j]
  ⑥ &(&q[i][0])[j]

c) Using Pointer q to Express Element of Two-dimensional Array

After the assignment
q=p; or q=&p[0];
there are several forms which use pointer q to express element of two-dimensional array:
①   *q[i]+j
②   *(q+i)+j
③   &q[i][0]+j
④   q[i][j]
⑤   *(q+i)[j]
⑥   (&q[i][0])[j]

d) Application Example 7

main()
{
  int a[3][4]= {1,2,3,4,5,6,7,8,9,10,11,12}, *p[3],**q,i,j;
  for(i=0;i<3;i++)
    p[i]=&a[i][0];
  q=p;
  for(i=0;i<3;i++)
    for(j=0;j<4;j++)
      printf("%d ",*(*(q+i)+j));
}

In this application, i and j are used to control cycle index. During the cycle process, the value of q is not changed, pointer q is not moving.

Example 8

main()
{
  int a[3][4]= {1,2,3,4,5,6,7,8,9,10,11,12}, *p[3],**q,i,j;
  for(i=0;i<3;i++)
    p[i]=&a[i][0];
  q=p;
  for(i=0;i<3;i++)
    for(j=0;j<4;j++)
      printf("%d ",(*q+i)+j));
      print("%d ",(*q+i)+j));
    q++;
    printf("\n");
}

In this application, i and j are used to control cycle index. During the outside loop process, the value of q is changed, pointer q is moving. After q++, q points to the next element of the pointer array p, by reference a pointer of the pointer array to access elements of a corresponding row of the two-dimensional array.

4) 4) The Pointer Variables Which Point to the Pointer Array

The pointer variable which points to the pointer array is defined as:
int a[3][4], *p[3], **q;
for(i=0;i<3;i++)
  p[i]=&a[i][0];

a) Assignment

Pointer q points to element zero of the pointer array, the correct assignment is:
q=p; or q=&p[0];
the wrong assignment is: q=a;

b) Using Pointer q to Express Element Address of Two-dimensional Array

After the assignment
IV. CONCLUSION

Using pointer to access the element of two-dimensional array, the key is to clear defined pointer points to the element or points to the one-dimensional array. The pointer which points to element is assigned a element address, the pointer which points to the one-dimensional array is assigned a row address. After we correctly assign value to pointer and correctly express element of two-dimensional array, we can smoothly use pointer to access the element of two-dimensional array.

REFERENCES


