C

Arrays and pointer arithmetic

Patrick Stiller

November 15, 2014

Introduction

An array is a fixed-length data structure that can contain multiple objects of the same type.

Code structure

To decleare an array use following structure:

```
1 // Structure
2
3 dataType arrayname[size];
4
5 // Examples
6
7 int alpha[5]; // Array with 5 int variables
8 char beta[6]; // Array with 6 char variables
```

Simple illustration

```
1  int alpha[4];
```

The first element of the array is decleared as nameOfArray[0].

variable	value of variable	address(hex)
alpha[0]	first element	FFFC
alpha[1]	second element	FFFD
alpha[2]	third element	FFFE
alpha[3]	fourth element	FFFF

Access to the values

To get access to the values of the array ,you need to use the index of the array.

```
1  int alpha[4];
2  int anInt;
3
4  // nameOfArray[index]
5  alpha[0]=2;
6  alpha[1]=3;
7  alpha[2]=1000;
8  alpha[3]=450;
9  anInt = alpha[2];
```

Range excess

- Accessing an index out of range can result in reading arbitrary memory or a segmentation fault.
- Caution: In an array with n elements , the highest allowed index is : n-1
- Always check if the index is valid before accessing an array.

```
1  int alpha[3];
2
3
4  Warning!!! range excess
5  alpha[4] = 3;
```

Example: range access

```
1  /* Manual access to array alpha */
2  int alpha[3];
3  int allowedIndex = 2; // n-1 | n = 3
4  int index;
5  int value =4;
6
7  scanf("%d",&index);
8  if(index < 0) || (index > allowedIndex){
9   printf("Invalid Index");
10 }
11 else{
12  alpha[index] = value;
13 }
```

Iterate

Advantage of arrays : it is easy to iterate.

```
1  int alpha[4];
2
3  //input
4  alpha[0] = 2;
5  alpha[1] = 3;
6  alpha[2] = 4;
7  alpha[3] = 1;
8
9  int index;
10
11 //output
12 for(index=0; index < 4; index = index + 1){
13   printf("\n %d",alpha[index]);
14 }
```

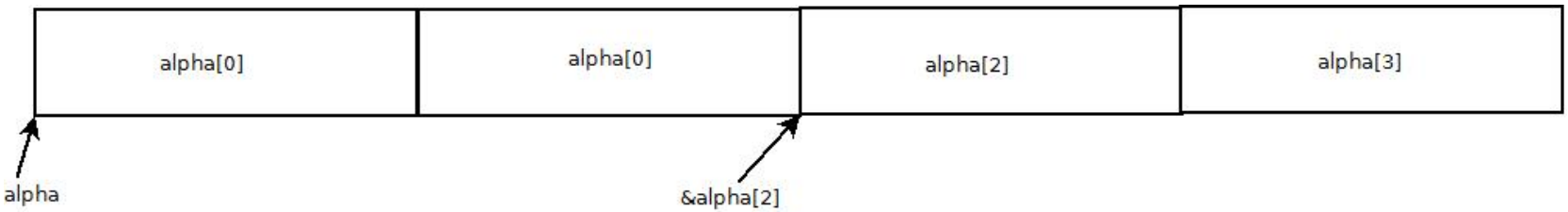
Introduction

A pointer points to an element of the array if you use the address-operator(&).

```
1 int alpha[5];
2 int* pointerToInt;
3
4 pointerToInt = &alpha[i-1];
5 // pointer points to i-th element of the array
6
```

Equivalence between array and pointer notation

The name of an array is a constant pointer to the first element of the array.



So we have two equivalent statements for the first element of the array.

```
1 int alpha[4];
2 int* firstElement;
3
4 // the two following statements are equivalent
5 firstElement = alpha; // equivalent
6 firstElement = &alpha[0] ; // equivalent
```

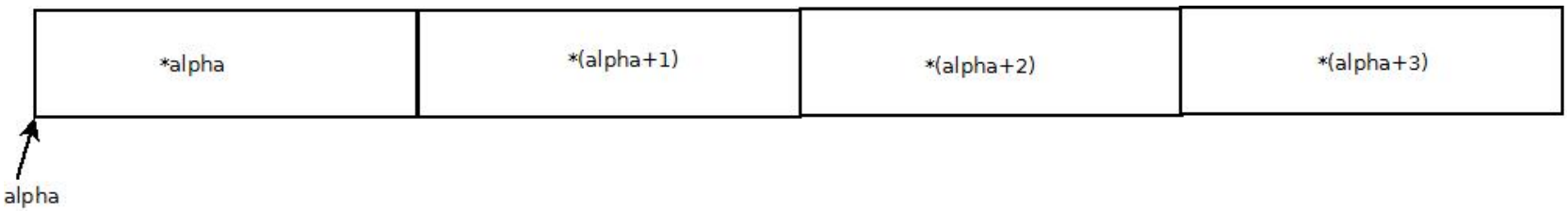
Array in pointer notation

If alpha is a pointer. You can transform :

```
1 *(alpha+i)
```

into:

```
1  alpha[i]
```



Picture: Array in pointer notation

Caution!

if you increment a pointer you do not get a pointer pointing to the next byte but to sizeof(pointedType) bytes later.

```
1 long foo;
2 long* pointer = &foo;
3 printf("pointer is at %p,
4 pointer+1 is at %p\n", pointer, pointer+1);
```

Comparison between two arrays

The comparison between two arrays is not possible. So you can't check:

```
1  int alpha[4];
2  int beta[4];
3
4  if(alpha==beta}
5  {
6  printf(" alpha and beta are similar");
7  }
```

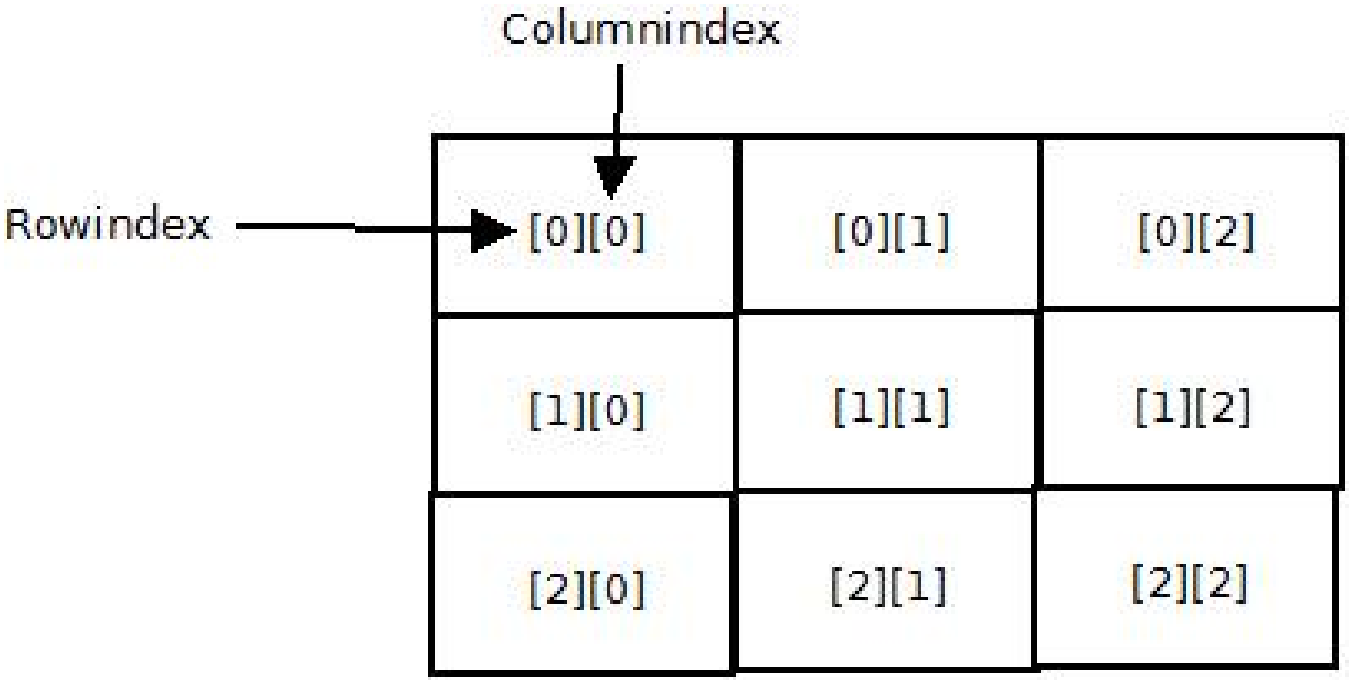
Because the name of an array is equivalent to the address of the first element of the array. So the program will test the equality of the two addresses.

Code Structure

To decleare an multidimensional Array use following structure.

```
1 dataType  nameOfArray [numberOfRows][numberOfcolumns];
2
3 // example
4
5 int alpha [3][3];
6
7 // an array with 3 rows and 3 columns
```

Simple Illustration

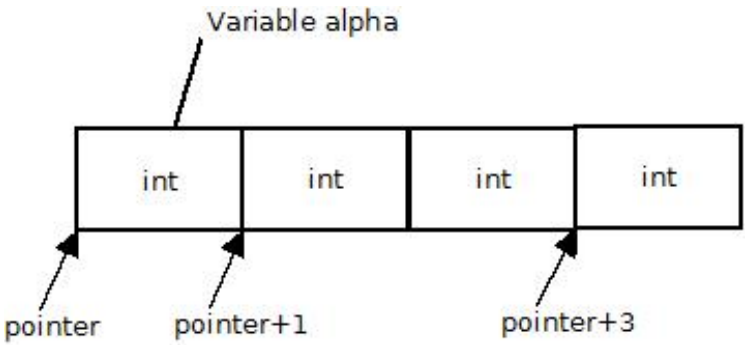


Equality

```
1   int* ptr1 = NULL;
2   int* ptr2 = NULL;
3   int integer1;
4   int integer2;
5   if(ptr1 == ptr2){
6    printf("EQUAL");
7     }
8
9   ptr1 = &integer1;
10  prt2 = &integer2;
11
12  if(ptr1 != ptr2){
13   printf("UNEQUAL");
14  }
15
```

Addition

```
1  int alpha;
2  int* pointer;
3
4  pointer = &alpha ;
```



Subtraktion is also possible. In the same way.