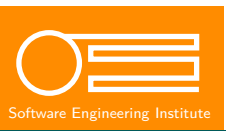
Introduction to C Programming Language
Pointer and Array
Lecture 10

Min Zhang

zhangmin@sei.ecnu.edu.cn

2020.12.07



10:00-11:40, Monday, Room 319
Software Engineering Institute, East China Normal University

Today's content

☞ Pointer & Array
(指针和他的好朋友们)

Recall the definition of Pointer

Pointer

A pointer is a variable that contains the address of some variable.

0	0	0	0	0	0	0	1	0x0000000000000000	`short int a;`
0	0	0	0	0	0	0	0	0x0000000000000001	`a=1;`
0	0	0	0	0	0	0	0	0x0000000000000002	`int *b;`
0	0	0	0	0	0	0	0	0x0000000000000003	`b=&a;`
0	0	0	0	0	0	0	0	0x0000000000000004	
0	0	0	0	0	0	0	0	0x0000000000000005	
0	0	0	0	0	0	0	0	0x0000000000000006	
0	0	0	0	0	0	0	0	0x0000000000000007	
0	0	0	0	0	0	0	0	0x0000000000000008	
0	0	0	0	0	0	0	0	0x0000000000000009	

Recall: Pointers as function arguments

```
1 void swap(int *,int *); // if you declare a function
2
3 void swap(int *i, int *j){ // if you define a function
4   int tmp=*j;
5   *j=*i;
6   *i=tmp;
7 }
```

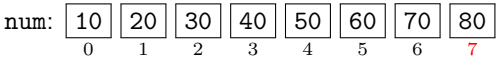
How to call a function whose arguments are pointers?

```
1 int a=2;
2 int b=3;
3 swap(&a,&b);
```

Arguments must be address!!!

Pointer and Array

Code: `int num[8];`
The size of `num` is 8.



Code: `int *ptr=num;`
`ptr` points to the place where `num[0]` is stored!

Code: `int *ptr1=num+1; // not num[1]`
`ptr1` points to the place where `num[1]` is stored!

The essence of an array name

What do you think the essence of an array name from the previous example?

An array name is essentially **a pointer**!

What does that mean?

1. You do not need operator & on it when you want to have the beginning address of the array.
2. You can use an array name as how pointers are used

Details will be given later.

Array as function argument: an example

```
1  void toUpper(char str[]){
2    int i=0;
3    while(str[i]!='\0'){
4      if(str[i]>='a'&&str[i]<='z') str[i]=str[i]-32;
5      i++;
6    }
7  }
8  int main(){
9   char hw[20]="hello world";
10  toUpper(hw); // characters in hw will be accessed
11  puts(hw);   // characters in hw have been capitalized
12  return 0;
13 }
```

What is the mechanism behind the code?

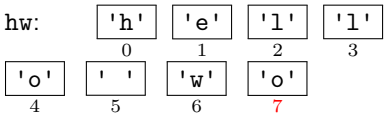
Array as function argument: an example

```
1  void toUpper(char str[]){
2  int i=0;
3  while(str[i]!='\0'){
4  if(str[i]>='a'&&str[i]<='z') str[i]=str[i]-32;
5  i++;
6  }
7  }
8  int main(){
9  char hw[20]="hello world";
10 toUpper(hw); // characters in hw will be accessed
11 puts(hw);   // characters in hw have been capitalized
12 return 0;
13 }
```

Array as function argument: an example

```
void toUpper(char str[]){
  int i=0;
  while(str[i]!='\0'){
    if(str[i]>='a'&&str[i]<='z') str[i]=str[i]-32;
    i++;
  }
}
```

```
int main(){
  char hw[20]="hello world";
  toUpper(hw);
  puts(hw);
  return 0;
}
```



What we learn: array names are pointers!

Use pointers to access array elements: an example

Now that array names are pointers, we can use them as how we use pointers!

```
1 int main(){
2   int nums[10],i;
3   int *ptr=nums;
4   for(i=0;i<10;i++){
5     nums[i]=0; // initialize array nums with 0
6   }
7 }
```

Use pointers to access array elements: an example

Now that array names are pointers, we can use them as how we use pointers!

```c
int main(){
  int nums[10],i;
  int *ptr=nums;
  for(i=0;i<10;i++){
    *(ptr+i)=0; // same
  }
}
```

Use pointers to access array elements: an example

Now that array names are pointers, we can use them as how we use pointers!

```c
int main(){
  int nums[10],i;
  int *ptr=nums;
  for(i=0;i<10;i++){
    ptr[i]=0; // you can even write in this way
  }
}
```

Use pointers to access array elements: an example

Now that array names are pointers, we can use them as how we use pointers!

```
1 int main(){
2   int nums[10],i;
3   int *ptr=nums;
4   for(i=0;i<10;i++){
5     *(nums+i)=0; // you can even write in this way
6   }
7 }
```

Use pointers to access array elements: an example

Now that array names are pointers, we can use them as how we use pointers!

```
1 void toUpper(char *str){
2     int i=0;
3     while(str[i]!='\0'){
4         if(str[i]>='a'&&str[i]<='z') str[i]=str[i]-32;
5         i++;
6     }
7 }
```

What do you learn from this example

Pointers and array names seem to be same!

YES, they are almost the same except that
the value of array names cannot be modified,
but the one of pointers can!

```
int main(){
  int nums1[10],nums2[20];
  int *ptr=nums1; // OK
  ptr=nums2; // you can modify ptr to point to nums2
  nums1=ptr; // you cannot modify nums1
  for(i=0;i<10;i++){
    *(nums+i)=0; // you can even write in this way
  }
}
```

Animation

nums1:

10	20	30	40	50	60	70	80	...
0	1	2	3	4	5	6	7	

nums2:

11	22	33	44	55	66	77	88	...
0	1	2	3	4	5	6	7	

```
ptr=nums1;
```

```
ptr=nums2;
```

```
nums1=nums2;
```
Bad

Address arithmetic

Look at our previous example:

```
1 void toUpper(char *str){
2   int i=0;
3   while(str[i]!='\0'){
4   if(str[i]>='a'&&str[i]<='z') str[i]=str[i]-32;
5     i++;
6   }
7 }
```

Why do we have to always use `i`?

Address arithmetic

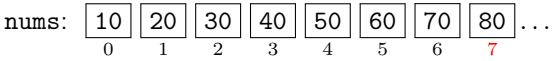
Look at our previous example:

```
1 void toUpper(char *str){
2   int i=0;
3   while(str[i]!='\0'){
4     if(str[i]>='a'&&str[i]<='z') str[i]=str[i]-32;
5     i++;
6   }
7 }
```

Why do we have to always use `i`?

Address arithmetic

We want to operate **directly on the pointer** to access all the element in an array!




```
int *ptr=nums;
ptr++;
ptr++;
ptr++;
ptr++;
ptr++;
ptr++;
ptr++;
```

Address arithmetic

A simpler solution:

```
1 void toUpper(char *str){
2   while(*str!='\0'){
3     if(*str>='a'&&*str<='z') *str=*str-32;
4     str++;
5   }
6 }
```

The principle of address arithmetic

Consider: **What does it mean by `ptr++`?**

We know that: `ptr=ptr+1;`

Suppose that the value of `ptr` is `0`, guess the value of `ptr` now: ??1

The principle of address arithmetic

Suppose that

```
1 int nums[20];
2 int *ptr = nums;
3 ptr++;
```

Suppose that the value of `ptr` is 0, guess the value of `ptr` now: ??4

Why?

Here is the answer:
When `i` is an integer variable, after `ptr=ptr+i`,
the value of `ptr` is: **`ptr + i * sizeof(type of ptr)`**
Note: function `sizeof` returns the number of bytes of a type

The principle of address arithmetic: example

```
1 int nums[20];
2 int *ptr = nums;
3 ptr++;
```

The value of `ptr`: `ptr+1*sizeof(int)`, `sizeof(int)` is 4.

```
1 char str[20];
2 char *ptr = str;
3 ptr++;
```

The value of `ptr`: `ptr+1*sizeof(char)`, `sizeof(int)` is 1.

Address arithmetic: subtraction of two pointers

When we do subtraction of **two pointers**, e.g. `ptr1-ptr2`, the result is **(ptr1-ptr2)/sizeof(ptr1)**.

```
1 int nums[10];
2 int *ptr=nums;
3 printf("%p\n",ptr);
4 int *ptr2=nums+10;
5 printf("%p,%p,%ld\n",ptr2,ptr1,ptr2-ptr);
```

Output: `0x7ffe8528f188,0x7ffe8528f160,10`