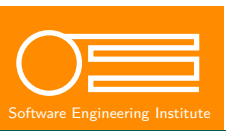
Introduction to C Programming Language
Pointer and Array
Lecture 12

Min Zhang

zhangmin@sei.ecnu.edu.cn

2020.12.14



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

Today's content

☞ **Pointer and Array**

Recall the relation between pointer and array name

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!

A string constant

```
"Hello world"
"Good morning"
"Zhang San"
"2017-12-11"
```

A string is essentially an array of characters!

When you call `printf("Hello world");`, the beginning address of the array storing "Hello world" is passed to `printf`.

```
char hw[]="Hello world"; // declare an array storing the string
char *hw="Hello world"; // declare a character pointer storing the beginning address of the array
    storing the string.
```

Character pointer as function argument

Recall the exercise: getInt,

传过来的字符数组

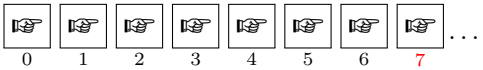
结果保存在 nums 里.

```
1 void getInt(char * str, int *nums){
2   while(*str!='\0'){ // check if the string reaches the end ]
3     if(*str>='0'&&*str<='9'){
4       *nums=*str-'0'; // store the current number to the integer array
5       nums++; // move to next empty cell in the integer array
6     }
7     str++; // move to the next character
8   }
9 }
```

Pointer array (指针数组)

```
type *ptrs[N]; // define N pointers as an array (保存了N个指针变量)
```

ptrs is the array name

ptrs: 


```
1 char* names[4]={"Zhang San", "Li Si", "Wang Wu", "Zhao Liu"};
2 for(i=0;i<4;i++)
3   printf("%s",names[i]); // or *(names+i)
```

Exercise: sort the strings and print them out in lexical order

Multi-dimensional arrays

Back to the previous example:

```
char* names[4]={"Zhang San", "Li Si", "Wang Wu", "Zhao Liu"};
```

Remember that: each string is an array. There are four arrays
The four arrays compose an array, which is called *array of arrays*, that is, two-dimensional array.

```
char names[4][10]={"Zhang San", "Li Si", "Wang Wu", "Zhao Liu"};
```

Another example:

```
int dates[2][13]={
  {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
  {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};
```

Usage: `dates[i][j]`: the `j`th element in the `i`th array.

The essence of two-dimensional array

We know that **an 1-d array name is essentially a pointer**.

```
int dates[2][13]={
  {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
  {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};
```

E.G. `dates[0]` is a pointer, `dates[1]` is a pointer.

Thus, **the array name of a 2-d array is a pointer, pointing to the first address of the first 1-d array.**

Declaration of a pointer to pointers

```
type **ptr;
```

Remember: `ptr` **points to a location storing a pointer.**

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	`short 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	

Definition of Pointer to pointers

Pointer

A pointer to pointers is a variable that contains the address of some **pointer(s)**.

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	short int *b;
0	0	0	0	0	0	0	0	0x0000000000000003	b=&a;
.	.	.	.	.	.	.	.	...	
0	0	0	0	0	0	0	0	0x0000000000000005	
0	0	0	0	0	0	0	0	0x0000000000000009	
0	0	0	0	0	0	1	0	0x000000000000000A	short int **c;
0	0	0	0	0	0	0	0	0x000000000000000B	c=&b;
.	.	.	.	.	.	.	.	...	
0	0	0	0	0	0	0	0	0x0000000000000011	

What do we use pointer to pointers

Pointer to pointers and 2-d arrays are good friends!

```
1 char names[4][10]={"Zhang San", "Li Si", "Wang Wu", "Zhao Liu"};
2 char *ptr[4]; // declare a pointer array
3 int i;
4 for(i=0;i<4;i++){
5   ptr[i]=names[i]; // let ptr point to names
6   printf("%s\n",*(ptr+i)); // or ptr[i], remember ptr[i] is a pointer
7 }
```

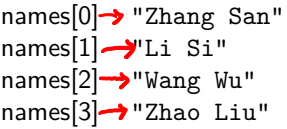
Now that `ptr` is the almost the same as `names`, why don't we just use `names` directly? Consider the following problem:

String sorting

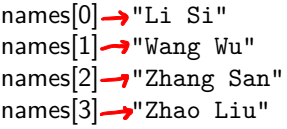
Define a function which sorts the names in `names`.

What do we do?

Before sorting

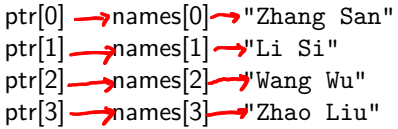


After sorting

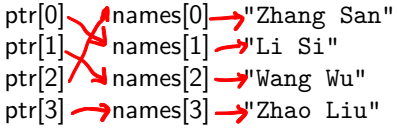


copying string is expensive!

Before sorting



After sorting



Not necessary to copy string!

Code

```
char names[4][10]={"Zhang San", "Li Si", "Wang Wu", "Zhao Liu"};
char *ptr[4]; // declare a pointer array
int i,j;
for(i=0;i<4;i++){ptr[i]=names[i];} // let ptr point to names
char *tmpPtr;
for(i=0;i<4;i++){
  for(j=i+1;j<4;j++){
    if(strcmp(ptr[i],ptr[j])>0){
      tmpPtr=ptr[i]; ptr[i]=ptr[j]; ptr[j]=tmpPtr;}
  }
}
for(i=0;i<4;i++){
  printf("%s\n",*(ptr+i));} // or ptr[i], remember ptr[i] is a pointer
```

What happens inside

Another problem

Write a function for string sorting!

```
1  char names[4][10]={"Zhang San", "Li Si", "Wang Wu", "Zhao Liu"};
2  char *ptr[4]; // declare a pointer array
3  int i,j;
4  for(i=0;i<4;i++){ptr[i]=names[i];} // let ptr point to names
5  char *tmpPtr;
6  for(i=0;i<4;i++){
7    for(j=i+1;j<4;j++){
8      if(strcmp(ptr[i],ptr[j])>0){
9        tmpPtr=ptr[i]; ptr[i]=ptr[j]; ptr[j]=tmpPtr;}}}
10 for(i=0;i<4;i++){
11   printf("%s\n",*(ptr+i));} // or ptr[i], remember ptr[i] is a pointer
```

Another problem

Write a function for string sorting!

```
1  void sortString(char *str[], int n){ // notice the parameter *ptr[]
2    char *tmpPtr;
3    int i,j;
4    for(i=0;i<n;i++){
5      for(j=i+1;j<n;j++){
6        if(strcmp(str[i],str[j])>0){
7          tmpPtr=str[i]; str[i]=str[j]; str[j]=tmpPtr;}}}
8  }
9
10 sortString(ptr,4); // the way of calling the function
```

2-d array as function argument

The following declarations are equivalent. The first argument should be a 2-d array. **The first value in the array is ignored by compilers**

1. `void sortString(char str[4][10], int n)`
2. `void sortString(char str[][10], int n)`
3. `void sortString(char (*str)[10], int n)`

It is different from: `void sortString(char *str[10], int n)`
Because **[] has a higher precedence than ***
`char *str[10]` is a pointer arrays.

Because numbers in array is **free**, the above declaration is equivalent to:
`void sortString(char *str[], int n)`, or
`void sortString(char **str, int n)`

Differences (cont.)

1. `void sortString1(char str[][10], int n)`

   The first argument should be an array name of a 2-d array

```
char strs[2][10]={"abc","def"};
sortString1(strs, 2); // call sorting
char **ppstr=strs; //bad
char *pstr=&strs[0][0];
sortString1(ppstr,2);//bad too
```

2. `void sortString2(char **str, int n)`

   The first argument should be a pointer (or an array name) to a pointer array

```
char *strs1[10]={"abc","def"};
sortString2(strs1, 2); // call sorting
char **ppstr=strs1; //good
sortString2(ppstr, 2); //good
sortString2(strs,2)// bad
```

Differences between pointer to pointers and 2-d array name

指针

```
1 char name[]="Zhang San";
2 char *pname=name;
```

=维数组

```
1 char name[][20]=
2     {"Zhang San","Li Si"};
3 char *pname=name[0];//OK
4 char *pname=name; // bad
5 printf("%s",name[1]);//?
6 printf("%s",pname+20);//?
```

一维数组地址

pointer to pointers

双重指针

```
1 char *names[4];
2 (char *)*pnames=names;
```

一个指针，保存了另外一个指针变量的地址。

- **A 2-d array is a pointer with length information**. The length is the size of the types of each 1-d array.
- **A pointer to pointers is a pointer with length being 8, which is the size of address**

Arguments of programs

指针数组

```
1 int main(int argc, char *argv[]){
2   int i=0;
3   while(i<argc){
4     printf("%s\n",argv[i++]);
5   }
6   return 0;
7 }
```

- argc: the number of arguments (including the command)
- argv: a pointer array with each pointing to a string in the arguments.

Exercise

Write a program named `convert` to convert its arguments into low-case or upper-case according to its option

- Command: `convert -l ABC`
  Result: `abc`
- Command: `convert -u abc`
  Result: `ABC`