

Pointers and Arrays

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

Pointers and Address

- A typical machine has **an array of consecutively numbered or addressed memory cells** that may be manipulated individually or in contiguous groups. For examples,
 - any byte can be a **char**
 - a pair of one-byte cells can be treated as a **short** integer
 - four adjacent bytes form a `long`
- The unary operator **&** gives the address of an object.
- The unary operator **&** only applies to objects in memory: variables and array elements. It cannot be applied to expressions, constants or register variables.
- The unary operator ***** is the *indirection or dereferencing* operator which can access the object an pointer points to when applied to a pointer.

```
1  #include <stdio.h>
2
3  int main() {
4      char ch = 'a';
5      int x = 123, arr[20] = {20, 19, 18};
6      float f = 3.14f;
7      double d = 2.17;
8
9      char *p_ch = &ch;
10     int *p_x = &x, *p_arr = arr;
11     float *p_f = &f;
12     double *p_d = &d;
13
14     printf("ch = %c\n", *p_ch);
15     printf("x = %d\n", *p_x);
16     printf("arr[0] = %d\n", *(p_arr + 0));
17     printf("f = %.2f\n", *p_f);
18     printf("d = %.2f\n", *p_d);
19
20     return 0;
21 }
```

- **int *p_x** says that the expression ***p_x** is an int and ***p_x** can occur in any context where **x** could.

```
1 *p_x = *p_x + 10; // increment *p_x by 10;
2 int y = *p_x + 1;
3 ++*p_x;
4 (*p_x)++; // parentheses is necessary because unary operators associate right to left
```

Pointers and Function Arguments

- C passes arguments to functions by value. Changing the arguments in the functions only changes the copies of objects.

```
1 #include <stdio.h>
2
3 void swap(int a, int b) {
4     int t = a;
5     a = b;
6     b = t;
7 }
8
9 int main() {
10     int x = 1, y = 2;
11     swap(x, y);
12     printf("x = %d, y = %d\n", x, y);
13
14     return 0;
15 }
16
17 /** The result is
18 x = 1, y = 2
19 */
```

- In order to access and change the objects, we can use pointers.

```
1 #include <stdio.h>
2
3 void swap(int *a, int *b) {
4     int t = *a;
5     *a = *b;
6     *b = t;
7 }
8
9 int main() {
10     int x = 1, y = 2;
11     swap(&x, &y);
12     printf("x = %d, y = %d\n", x, y);
13
14     return 0;
15 }
16
17 /** The result is
```

```
18  x = 2, y = 1
19  */
```

```
1  #include <stdio.h>
2
3  int getint(int *px) {
4      return scanf("%d", px);
5  }
6
7  int main() {
8      int x;
9
10     getint(&x);
11     printf("x = %d\n", x);
12
13     return 0;
14 }
15
16 /** The result is
17 2333
18 x = 2333
19 */
```

Pointers and Arrays

Suppose that we have an array **a[10]** and a pointer named **pa** defined as follow.

```
1  int a[10];
2  int *pa;
```

There are some relationships between arrays and pointers.

- The name of an array is the **address of first element** in the array and it can be assigned to a pointer.

```
1  pa = a; // pa points to the first element of a
```

- **a[i]** represents the $(i + 1)^{th}$ element in the array (counted from zero).

```
1  int x = a[3]; // The value of x is the same as the third element in a
2  pa = &a[0]; // which is equivalent to pa = a, pa points to the first element of a
```

- If **pa** points to a particular element in the array, then **pa + i** points *i* elements after **pa** and **pa - i** points *i* elements beyond **pa**. When **pa** points to the first element in the array, ***(pa + i)** which is equivalent to **a[i]** is the value of $(i + 1)^{th}$ element in the array.

```

1  pa = &a[3];
2  int a_7 = *(pa + 4); // the 7th element in a
3  int a_1 = *(pa - 2); // the second element in a
4
5  pa = a;
6  *(pa + 3) = 3; // the fourth element is changed to 3
7  *pa++; // the fifth element in a, changing the content of a pointer is ok

```

- If a pointer points to an array, it can be treated as the name of the array. Using this pointer with subscript is ok.

```

1  pa = a;
2
3  pa[2] = 2; // the third element in a is changed to 2

```

- A name of an array can be treated as a pointer, but we cannot change the object it points. It always points to the first element.

```

1  int a_3 = *(a + 3); // the value of a[3]
2  ++a; // error

```

- The name of an array can be passed to a function and it will be treated as a pointer.

```

1  #include <stdio.h>
2
3  #define LEN 8
4
5  void fun(int a[], int len) {
6      for (int i = 0; i < len; ++i)
7          printf("a[%d] = %d\n", i, *a++);
8  }
9
10 int main() {
11     int a[LEN] = {0, 1, 2, 3, 4, 5, 6, 7};
12
13     fun(a, LEN);
14
15     return 0;
16 }
17
18 /** The result is
19 a[0] = 0
20 a[1] = 1
21 a[2] = 2
22 a[3] = 3
23 a[4] = 4
24 a[5] = 5
25 a[6] = 6
26 a[7] = 7
27 */

```

- Using pointers is more faster than array but not easily to understand. And sometimes it will bring some troubles if you don't use pointers correctly.