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 ***texttlong***
- 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.

```c
#include <stdio.h>

int main() {
    char ch = 'a';
    int x = 123, arr[20] = {20, 19, 18};
    float f = 3.14f;
    double d = 2.17;

    char *p_ch = &ch;
    int *p_x = &x, *p_arr = arr;
    float *p_f = &f;
    double *p_d = &d;

    printf("ch = %c\n", *p_ch);
    printf("x = %d\n", *p_x);
    printf("arr[0] = %d\n", *(p_arr + 0));
    printf("f = %.2f\n", *p_f);
    printf("d = %.2f\n", *p_d);

    return 0;
}
```

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

```c
*p_x = *p_x + 10; // increment *p_x by 10;
int y = *p_x + 1;
++*p_x;
(*p_x)++; // parentheses is neccessary 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.

```c
#include <stdio.h>

void swap(int a, int b) {
    int t = a;
    a = b;
    b = t;
}

int main() {
    int x = 1, y = 2;
    swap(x, y);
    printf("x = %d, y = %d\n", x, y);

    return 0;
}

/** The result is
x = 1, y = 2
*/
```

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

```c
#include <stdio.h>

void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int main() {
    int x = 1, y = 2;
    swap(&x, &y);
    printf("x = %d, y = %d\n", x, y);

    return 0;
}

/** The result is
```

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

```c
#include <stdio.h>

int getint(int *px) {
    return scanf("%d", px);
}

int main() {
    int x;

    getint(&x);
    printf("x = %d\n", x);

    return 0;
}

/** The result is
2333
x = 2333
*/
```

Pointers and Arrays

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

```c
int a[10];
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.

```c
pa = a; // pa points to the first elemtnt of a
```

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

```c
int x = a[3]; // The value of x is the same as the third element in a
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.

```c
pa = &a[3];
int a_7 = *(pa + 4); // the 7th element in a
int a_1 = *(pa - 2); // the second element in a

pa = a;
*(pa + 3) = 3; // the fourth element is changed to 3
*pa++; // the fifth eleement 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.

```c
pa = a;

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.

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

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

```c
#include <stdio.h>

#define LEN 8

void fun(int a[], int len) {
    for (int i = 0; i < len; ++i)
        printf("a[%d] = %d\n", i, *a++);
}

int main() {
    int a[LEN] = {0, 1, 2, 3, 4, 5, 6, 7};

    fun(a, LEN);

    return 0;
}

/** The result is
a[0] = 0
a[1] = 1
a[2] = 2
a[3] = 3
a[4] = 4
a[5] = 5
a[6] = 6
a[7] = 7
*/
```

- 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.