L7 - Pointers

November 1, 2019

0.1 Pointers

We can view a computer's memory as a collection of consecutively-numbered cells, each holding an 8-bit *byte*. A *char* (one ASCII standard character) can be stored in a single byte. A 32-bit integer is stored in 4 adjacent cells. When we make a variable in C, the variable's name is a symbolic name for this group of bytes in the computer's memory.

Each cell is numbered with an *address*. A **pointer** is a variable that contains another variable's address.

"I don't know what the value you're looking for is, but I know where it is."

```
In [ ]: int *p; // the address of p;
        int* p; // equivalent; the type of p is pointer to int
```

Note that this has a type of "pointer to int"; it is *not* an integer assigned to *p. This type refers to the object pointed at, not to the pointer itself.

0.1.1 Address-Of Operator and Dereference Operator

&: the *reference* or *address-of* operator. Returns the address where the variable is stored.
*: the *dereference* or *content-of* operator. Returns the value of the variable at the address.

These operators have higher precedence than arithmetic, comparison, and logical operators. A pointer will be dereferenced before addition, for example.

Pointers can also point to other pointers.

```
In [3]: #include <stdio.h>
        #include <stdlib.h>

        void main(){

            int* p;
            int* q;

            int x = 1, y = 2, z = 8;

            p = &x; // assigns the address of x to p
            q = &z;

            printf("address: %p\n", p); // some hexadecimal garbage
```

```
        *p = 5; // the variable pointed to by p is assigned 5
                // x now has the value 5

        printf("x = %d; y = %d\n", x, y); // see?

        // now we can use *p anywhere we would have used x
        *p = *p + 10;

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

        // pointer to a pointer
        q = p;

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

        *q = 99;

        printf("*p = %d; *q = %d\n", *p, *q); // both pointers point to x

        *p = *q - 11;

        printf("*p = %d; *q = %d\n", *p, *q);

    }
```

```
address: 0x7ffef3fcc06c
x = 5; y = 2
x = 15; y = 2
x = 15; y = 2
*p = 99; *q = 99
*p = 88; *q = 88
```

```
In [6]: #include <stdio.h>
        #include <stdlib.h>

        int main(void)
        {
            int val = 33;
            int* ptr = NULL;

            printf("Initial ptr: %p\n\n", (void *)ptr);

            ptr = &val;

            printf("Address of val (&val) = %p\n", (void *)&val);
            printf("Address of val (ptr) = %p\n\n", ptr);
```

```
            printf("val = %d, *ptr = %d\n\n", val, *ptr);
            printf("address of ptr (&ptr) = %p\n", (void *)&ptr);

            return EXIT_SUCCESS;
        }
```

```
Initial ptr: (nil)

Address of val (&val) = 0x7ffd3daf1a2c
Address of val (ptr) = 0x7ffd3daf1a2c

val = 33, *ptr = 33

address of ptr (&ptr) = 0x7ffd3daf1a30
```