Chapter 7 - Pointers

```
// declaring variables
int x;       // simple integer variable
int * xPtr;  // xPtr is a pointer to an integer

// assigning something to a pointer variable
x = 12;     // store an integer in x
xPtr = &x;  // store the address of x in xPtr

 // print the value of x
printf( "x = %d\n", x );

// print the pointer xPtr using %p as the placeholder
printf("xPtr = %p\n", xPtr);

// print the value of x but using xPtr
// this is called "dereferencing" the pointer
printf( "x = %d\n", *xPtr );

// change the value of x to 25, using xPtr
*xPtr = 25;
```

How to read these symbols

```
&   "the address of . . ."
*    "is a pointer to. . ." when used in a variable declaration
*    "the contents of . . ." when used for dereferencing
```

One of the main reasons that we have to use pointers in C is that all parameters are pass-by-value. We can simulate pass-by-reference parameters by sending a pointer to a function.

Example

Write a function that squares its parameter and the change is communicated back to the calling function.

```
void square ( double *dPtr ) {
     *dPtr *= *dPtr;
or
     *dPtr = *dPtr  *  *dPtr;
      We would read this statement as:  "The contents of dPtr is changed to the
      contents of dPtr multiplied by the contents of dPtr."
}
```

Another way to write the square function is:

```
void square ( double *dPtr ) {
       double number = *dPtr;  // dereference the pointer
       number = number * number;
       *dPtr = number;
}
```

You can do arithmetic with pointers.

```
xPtr++;           //  increment the pointer by the number of bytes
                  //  it takes to store an int  (usually 4 bytes)
xPtr = xPtr + 5;  //   increase xPtr by # of bytes it takes to store 5 ints
xPtr = xPtr - 3;  //   decrease xPtr by # of bytes it takes to store 3 ints
xPtr--;           // decrease by 1 int of space (4 bytes)
```

What If?

What happens if you increase xPtr or decrease it and now it's pointing to some unknown thing?

```
int x;
int *xPtr;
x = 12;
xPtr = &x;
xPtr++;    //  what is xPtr pointing to?   we don't know
```

Can I print xPtr? with %p? yes

Can I print *xPtr? with %d? maybe.

If those 4 bytes belong to RAM that's mine, then yes. If those 4 bytes belong to someone else's program or to the operating system, then we get "segmentation fault".

Ever seen 4 bytes of memory? what do they look like? It's a bunch of zeros and ones.

0010 0000 0111 0101 1010 0010 0011 0111

Arrays with pointers.

The name of an array is a pointer to the first element of the array.

```
int array[ 10 ];
int * arrayPtr = &array[0];
//  store the odd numbers 1, 3, 5 … in the array
//  using arrayPtr
int i;
for ( i = 0; i < 10; i++ ) {
      *arrayPtr = i * 2  +  1;
      arrayPtr++;

}
```

You can't change the pointer that is associated with the name of the array. For example, **array++** is invalid.

Using const

const is a reserved word (key word) in C

If we define a variable or a parameter using the word const, it prevents the value from being changed.

```
const int NUMBER = 10;    //  now NUMBER cannot be changed by
                          //  any statement in the program
```

It's a syntax error to attempt to change the value of NUMBER.

We can use the word const in two different places when declaring pointers.

```
const int * numPtr;       numPtr is a pointer to a constant integer
                          1)  the pointer can be changed
                          2) the data stored at that memory location
                             cannot be changed

int * const anotherPtr;   anotherPtr is a constant pointer to integer data
                          1)  we cannot change the pointer
                          2) we can change the contents of the memory
                             location

int * xPtr;               xPtr is a non-constant pointer to non-constant data
```

const int * const totallyConstantPointer; totallyConstantPointer is a constant pointer to constant data. Neither the pointer nor the data to which it points can be changed.

Principle of Least Privilege - give a function the privileges that it must have to complete its assigned task… and nothing more.