Introduction to Pointers

Darin Brezeale

The University of Texas at Arlington

Addresses in Memory

Everything in memory has an address. C allows us to obtain the address that a variable is stored at. In fact, if you have used `scanf()` you have already done this, for example,

```
scanf("%d", &year);
```

Addresses in Memory

Preceding a variable name by an ampersand, known as the *address operator*, will return its address:

```
#include <stdio.h>

int main(void)
{
    int x;

    /* notice the format specifier in printf() for an
       address is %p                                    */
    printf("The address for the memory allocated to x is %p\n", &x);
}
```

produces

```
    The address where x will store its value is 0xbfc54a04
```

Number Systems

The following number systems are found in computer science:

Common Name	Base	Digits
binary	2	01
octal	8	01234567
decimal	10	0123456789
hexadecimal	16	0123456789ABCDEF

In the preceding example, we can tell that the number was in hexadecimal because it began with `0x`.

Pointers

A **pointer** is a variable whose contents are the address of another variable.

```
int num = 3;
int* numptr;  /* numptr is a pointer  */

printf("content of num is %d\n", num);
printf("address of num is %p\n", &num);

numptr = &num;  /* initialize numptr with the
                   address of num             */
printf("content of numptr is %p\n", numptr);
```

produces

```
content of num is 3
address of num is 0x7fffffcbba4
content of numptr is 0x7fffffcbba4
```

Pointers

Pointers allow us to modify locations in memory by prefixing an initialized pointer with an asterisk, known as the *dereference operator*.

```
int num = 3;
int* numptr = &num;

printf("content of num is %d\n", num);
printf("content pointed to by numptr is %d\n", *numptr);

/* we can use the pointer to change what it points to */
*numptr = 99;

printf("content of num is now %d\n", num);
```

produces

```
content of num is 3
content pointed to by numptr is 3
content of num is now 99
```

Pointers

We can use pointers in much the same way we do the variables that they point to.

```
int a = 3, b = 3; /* a and b start with equal values */
int* bptr = &b;   /* we'll modify b using a pointer */

a += 4;
*bptr += 4;
printf("a is %d, b is %d\n", a, b);

a++;
(*bptr)++; /* parentheses are necessary here to override
              the order of precedence                    */
printf("a is %d, b is %d\n", a, b);
```

produces

```
a is 7, b is 7
a is 8, b is 8
```

Pointer Variable Types

Pointers are variables and they have their own type.

Example:

```
int* numptr;
```

`numptr` has a type of `int*` or pointer-to-int and should be initialized to point to a variable of type `int`.

Incrementing Pointers

If `ptr` is a pointer, what does `ptr++` mean?

This increments the address in the pointer to the address of the next variable of the type pointed to by the pointer. Example:

```
double num = 99;       /* doubles use 8 bytes of memory */
double* ptr = &num;

printf("ptr stores the address %p\n", ptr);
ptr++;
printf("now ptr stores the address %p\n", ptr);
```

produces

```
ptr stores the address 0x7fff63993b00
now ptr stores the address 0x7fff63993b08
```

Pointers to Pointers

Pointers can contain the address of another pointer.

```
int num = 5;
int* numptr = &num;
int** ptr2 = &numptr;  /* notice the two asterisks */
```

Comparing Pointers

We need to differentiate between comparing the contents of pointers and the variables that pointers point to. To compare the addresses stored in pointers, use

```
if(numptr == valptr)
```

To compare the values of the variables that pointers point to, use

```
if(*numptr == *valptr)
```

Initializing Pointers to NULL

- When a pointer variable is created, its initial value is whatever is in its allocated memory location just like other variables.
- A pointer may be initialized with an address later in a program based upon certain conditions.
- Sometimes we wish to initially set the pointer to a value that later can be used to determine if the pointer was never assigned an address.
- The value we use for this is NULL (in uppercase).

Initializing Pointers to NULL

```
#include <stdio.h>

int main(void)
{
    int num = 3;
    int* numptr;

    numptr = NULL;

    if (numptr != NULL)
        printf("num is %d\n", *numptr);
    else
        printf("Oops.  numptr has a value of %p\n", numptr);
}
```

produces

```
Oops.  numptr has a value of 00000000
```

Pointers and Functions

Previously, we made function calls like this:

```
int x = 3;
int y;
y = do_something(x);
```

In this case, a copy of the variable's value are passed to the function in a process called *pass by value*.

Changes made to the copy do not affect the original value.

Pointers and Functions

Pointers allow us to use a process called *pass by reference*, in which we will be able to change the value of the original variable. We do this by passing the variable's address to the function.

Pointers and Functions

```
#include <stdio.h>

void tripleNum(int*); /* notice the function argument
                         has a type of int *          */
int main(void)
{
    int num = 8;
    printf("before the function call, num is %d\n", num);
    tripleNum(&num); /* pass address */
    printf("after the function call, num is %d\n", num);
}

void tripleNum(int* aptr)   /* pass by reference */
{
    *aptr = 3 * *aptr; /* first asterisk is for multiplication,
                          second is to dereference the pointer  */
}
```

produces

```
    before the function call, num is 8
    after the function call, num is 24
```

Arrays of Pointers

A pointer is a variable type and we can have an array of pointers just as we have had arrays of other variable types.

Arrays of Pointers

```
#include <stdio.h>

int main(void)
{
    int i;
    char* text[4] = {"some string",      /* text[0] */
                     "another string",   /* text[1] */
                     "word"};            /* text[2] */

    text[3] = "\nassign this way";

    for(i = 0; i < 4; i++)
        printf("%s\n", text[i]); /* note the %s format specifier */
}
```

```
some string
another string
word

assign this way
```