Introduction to Pointers

Darin Brezeale
Modified by Vassilis Athitsos

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);
```

Pointers

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

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

```
void increment_number(int * x)
{
  *x = *x + 1;
}

int main()
{
  int a = 10;
  increment_number(&a);
  printf("a = %d\n", a);
}
```

Using dereference operator

Using address operator

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`.

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* aptr)    /* pass by reference */
{
  *aptr = 3 * *aptr; // first asterisk is for multiplication
                     // second is to dereference the pointer */
}

int main(void)
{
  int num = 8;
  int* numptr = &num;
  printf ("before the function call, num is %d\n", num);
  tripleNum (numptr);
  printf ("after the function call, num is %d\n", num);
}
```

Produces:

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

Pointers and Functions

Just because a function has pointer arguments doesn't mean we must create pointer variables in the calling function. Instead, we can use the address operator to pass the address. In our previous example, we could have used:

```
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);
```

Returning a Pointer

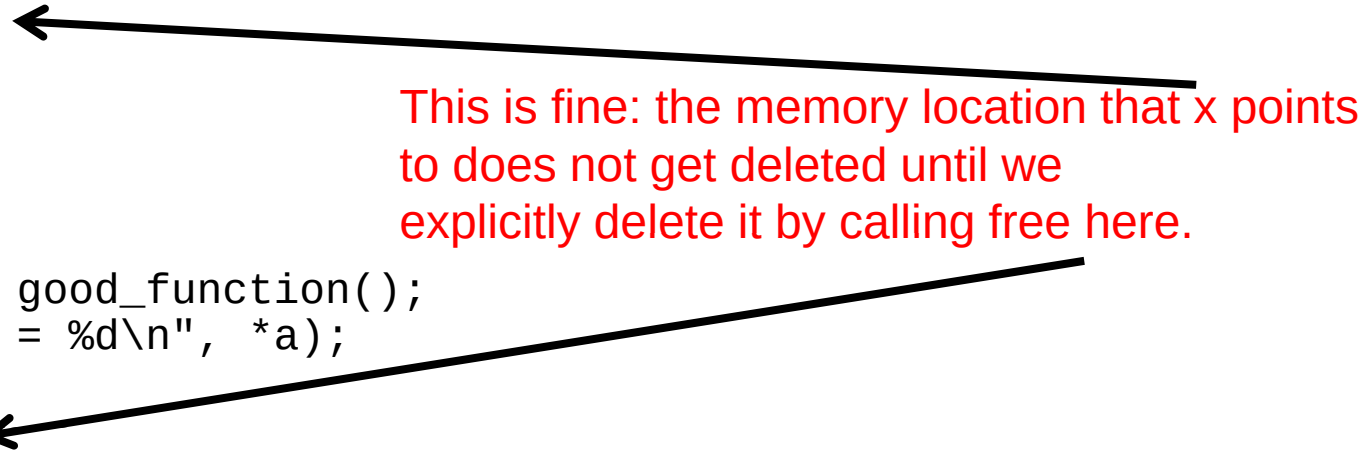
When you write a function that returns a pointer, you must make sure that the pointer points to a valid memory location.

Good example:

```
int * good_function()
{
  int * x = (int *) malloc(sizeof(int) * 1);
  *x = 5;
  return x;
}


int main()
{
  int * a = good_function();
  printf("a = %d\n", *a);

  free(a);
}
```



Returning a Pointer

When you write a function that returns a pointer, you must make sure that the pointer points to a valid memory location.

Bad example:

```
int * bad_function()
{
  int x = 5;
  return &x;
}

int main()
{
  int * a = bad_function();
  printf("a = %d\n", *a);
}
```

Illegal: the memory location corresponding to x gets deleted as soon as bad_function returns.