Pointers

ELEC1201 Lecture 6

Enrico Costanza

11 October 2013

Last edit: 11 October
Textbook + Links

• The material covered in this lecture is in chapter 7
• Please go through chapters 6 and 7 before the lab
• Please note that these slides contain links to example code, follow the links so that you can experiment with the code
Credits + feedback

• The quizzes in these slides are from http://www.cprogramming.com/

• The rest of the material is original, so feedback is welcome!
Pointers

• One of the most powerful features of C!

• Why use them?
  – Change a variable passed to a function (this also becomes a way to return values)
  – Be fast!
  – Dynamic allocation (more about this in a week or two)
What is a pointer?

• A variable containing the memory address of another variable
• A pointers *points to a variable (of a specific type)
• Declared as follows: `<type> * <ptrName>;`
What is a pointer?

• Example:

```c
int var = 3;
int * ptr;
ptr = &var;

*ptr = 44;
printf("*ptr: %d\n", *ptr);
```

• ‘&’ is the address-of operator

• The pointer can be used to access the variable data through the *indirection* operator ‘*’:

```c
*ptr = 44;
printf("*ptr: %d\n", *ptr);
```

• ptr1.c example
Declare and assign in one step

• You know you can do:
  int var = 5;
  and it’s the same as:
  int var;
  var = 5;

• You can also do:
  int * ptr = &var;
  this may be a bit confusing, but it is the same as:
  int * ptr;
  ptr = &var;
  /* DIFFERENT from *ptr = &var; !!! */
Binky Pointer Video

- From [http://cslibrary.stanford.edu/104/](http://cslibrary.stanford.edu/104/)

(ignore the "malloc" part of the story for now)
Pointers to pointers

• Remember: a pointer is itself a variable, so we can have a pointer that points to it

```c
int var = 3;
int * ptr = &var;
Int **ptrptr = &ptr;
```

• Why do we need double pointers? Examples later..
Quiz to check your understanding

Which of the following is the proper declaration of a pointer?

- A. int x;
- B. int &x;
- C. ptr x;
- D. int *x;
Quiz to check your understanding

- Which of the following gives the memory address of integer variable a?
  - A. *a;
  - B. a;
  - C. &a;
  - D. address(a);
Quiz to check your understanding

- Which of the following gives the memory address of a variable pointed to by pointer a?
  - A. a;
  - B. *a;
  - C. &a;
  - D. address(a);
Quiz to check your understanding

• Which of the following gives the value stored at the address pointed to by pointer a?
  
  – A. a;
  – B. val(a);
  – C. *a;
  – D. &a;
Pointers as function arguments

• If you use a pointer as an argument to a function, you can change the data it points to

```c
void reset(int * ptr)
{
    *ptr = 0;
    return;
}
```

• `ptr2.c` & `ptr3.c` examples
Changing the pointer value?

• The example ptr3.c shows that the value of the pointer (as opposite to the value of the pointed variable) is copied into the function: modifications to it do not influence the caller function

• What if we want to manipulate a pointer from inside a function (so that the effects influence the caller)?
  
  – That's one case where double pointers are useful
Arrays & pointers

• The name of an array gets automatically converted to a pointer:

```c
int array[5];
array == &array[0] is true
```

• So you can use:

```c
int * ptr;
ptr = array; /* equivalent to ptr = &array[0]; */
```
Arrays & pointers (cont.)

• Because array elements are always stored in sequential memory locations, you can access the elements by increasing the pointer value!

• `int * ptr = array;`  
  `ptr` points to `array[0]`  
  `ptr+1` points to `array[1]`  
  `ptr+2` points to `array[2]`  
  etc..

• Example: `arrayptr1.c`
Arrays, pointers & functions

• Because array names get converted automatically to pointers, when we pass an array to a function we actually pass a pointer, so we can modify the array content

• Note that arrays of arrays do NOT get automatically converted to pointers to pointers (a.k.a. double pointers)
Quiz to check your understanding

Which of the following gives the memory address of the first element in array foo, an array with 100 elements?

- A. foo[0];
- B. foo;
- C. &foo[0];
- D. foo[1];

Please note that there was a typo on this slide when I showed in the lecture. It is now fixed, apologies!
Quiz to check your understanding

• What will this code print?
  - A. 1
  - B. 2
  - C. 3
  - D. none of these / cannot tell

```c
int func(int a)
{
    a = a + 5;
    return 2;
}
int main(void)
{
    int a=3, b=1;
    a = func(b);
    printf("%d", a);
}
```
Quiz to check your understanding

• What will this code print?
  – A. 1
  – B. 2
  – C. 3
  – D. none of these / cannot tell

```c
int func(int a){
a = a + 5;
return 2;
}
int main(void)
{
    int a=3, b=1;
a = func(b);
printf("%d", b)
```
Quiz to check your understanding

• What will this code print?
  – A. 1
  – B. 2
  – C. 3
  – D. none of these / cannot tell

```c
int func(int * a) {
    a = a + 5;
    return 2;
}
int main(void) {
    int a=3, b=1;
    a = func(b);
    printf("%d", b)
}
Quiz to check your understanding

• What will this code print?
  – A. 1
  – B. 2
  – C. 3
  – D. none of these / cannot tell

```c
int func(int * a) {
    a = a + 5;
    return 2;
}
int main(void) {
    int a=3, b=1;
    a = func(&b);
    printf("%d", b);
}
```
Quiz to check your understanding

What will this code print?

- A. 1
- B. 2
- C. 6
- D. none of these / cannot tell

```c
int func(int * a){
    *a = *a + 5;
    return 2;
}
int main(void){
    int a=3, b=1;
    a = func(&b);
    printf("%d", b);
}
```
Generic Pointers: void *

- The keyword `void` can be used when declaring functions to mean "no arguments" or "no return value".
- The same keyword `void` can also be used to declare a *generic pointer*.
- A generic pointer is a pointer to a variable of undefined type.
- Generally, this is only useful as a transient format, to pass a pointer around functions without specifying the type.
- We need to cast a generic pointer to a specific pointer type before we can dereference it (i.e. use it).
Summary

• Pointers are a very powerful feature of C

• A pointer can contain the address of another variable

• We can access (read and write) the pointed variable through the pointer – this way we can change the value of function arguments

• Pointers can be moved around through pointer arithmetic, useful with arrays