CS 2261: Media Device Architecture - Week 4, part 1.5

QUIZ 1

Everything to the side of the room as you enter.

Just need writing instrument and Buzz Card at your desk.

Overview

- Macro functions vs real functions
- More on Pointers

Suppose...

- We need the function max(a,b)

- We need it for several different types
  - ints
  - floats
  - unsigned
  - etc,

- Should we write a function for each?
  - int max(int a, int b)
  - float max(float a, float b)
  - etc.

The Macro Solution

- We can write a single macro which will work for different types!

```
#define max(a, b) a >= b ? a : b
int x = 7;
int y = 8;
float p = 78.6;
float q = 29.2;
```

A Macro Gotcha

```
#define SQUARE(x) ((x)*(x))

int x = 2;
int z = SQUARE(x++);

What's the correct answer?
z = x^2                 // 4
z = (x + 1)^2           // 9
z = x * (x + 1)         // 6
z = (x + 1) * (x + 2)  // 12

Even more importantly what has happened to the
value of x???
```

A Macro Gotcha

```
#define SQUARE(x) ((x)*(x))

int x = 2;
int z = SQUARE(x++);

What’s the correct answer?
z = x^2                // 4
z = (x + 1)^2          // 9
z = x * (x + 1)        // 6
z = (x + 1) * (x + 2)  // 12
```

Even more importantly what has happened to the value of x???

What happens?

```
/* Macro */

SQUARE(x)
.
.
.
SQUARE(x)
.
.
.
SQUARE(x)
```

```
/* Function */

square(x)
.
.
.
square(x)
.
.
.
square(x)
```

What happens?

```
/* Macro */

((x) * (x))
.
.
.
((x) * (x))
.
.
.
((x) * (x))
```

What happens?

```
/* Macro */

((x) * (x))
.
.
.
((x) * (x))
.
.
.
((x) * (x))
```

```
/* Function */

pass parameter(s)
call function
.
pass parameter(s)
call function
.
pass parameter(s)
call function
.
square function
return
```

What happens?

```
/* Macro */

((x) * (x))
.
.
.
((x) * (x))
.
.
.
((x) * (x))
```

```
/* Function */

pass parameter(s)
call function
.
pass parameter(s)
call function
.
pass parameter(s)
call function
.
square function
return
```

Macros vs. Functions

- Macros
  - Text substitution at Translation (compile) time
  - May have problems: e.g. square(x++)
  - Will work with different types due to operator overloading
    - floats, doubles, ints, …
  - Difficult to implement if complex

- Functions
  - Separate piece of code
  - Overhead of passing arguments and returning results via stack
  - Fixes ambiguity problems: e.g. square(x + y) or(x++)
  - Function optimizes for space. Why?

Macros vs Functions

- If the goal is not clearly optimization for speed or time but rather somewhere in-between it’s difficult to know exactly which choice is correct.
- In any event: Don’t try and outwit the compiler!
- A better algorithm is more of an improvement that trying to write tricky code!!!

Pointers as variables

```
// some static variables
int foo;
int *bar;
int **baz;

/*
 ^ a pointer to a pointer!?
 an integer pointer pointer
 an integer double pointer
*/

int main(){
  ...
}
```

Variables with static storage duration are initialized with a default value.

For numeric types, that value is 0 (signed or unsigned).

For pointers (of all kinds), the default is NULL. Dereferencing a NULL Pointer is like a Null Pointer Exceptions in Java

- In C, you might see a lot of "Segmentation fault (core dumped)" messages if you do this.

Pointers as variables

```
// some static variables
int foo;
int *bar;
int **baz;

int main(){
  ...
}
```

Variable Table

- These variables will be in the static section of memory. Let's just call that 0xF0, for example purposes only.

Name	Address	Value
foo	0xF0	0
bar	0xF4	NULL
baz	0xF8	NULL

Pointers as variables

```
// some static variables
int foo;
int *bar;
int **baz;

int main(){
  // set bar to address of foo
  bar = &foo;
}
```

Variable Table

- These variables will be in the static section of memory. Let's just call that 0xF0, for example purposes only.

Name	Address	Value
foo	0xF0	0
bar	0xF4	0xF0
baz	0xF8	NULL

Pointers as variables

```
// some static variables
int foo;
int *bar;
int **baz;

int main(){
  // set bar to address of foo
  bar = &foo;

  // set baz to address of bar
  baz = &bar;
}
```

Variable Table

- These variables will be in the static section of memory. Let's just call that 0xF0, for example purposes only.

Name	Address	Value
foo	0xF0	0
bar	0xF4	0xF0
baz	0xF8	0xF4

Pointers as variables

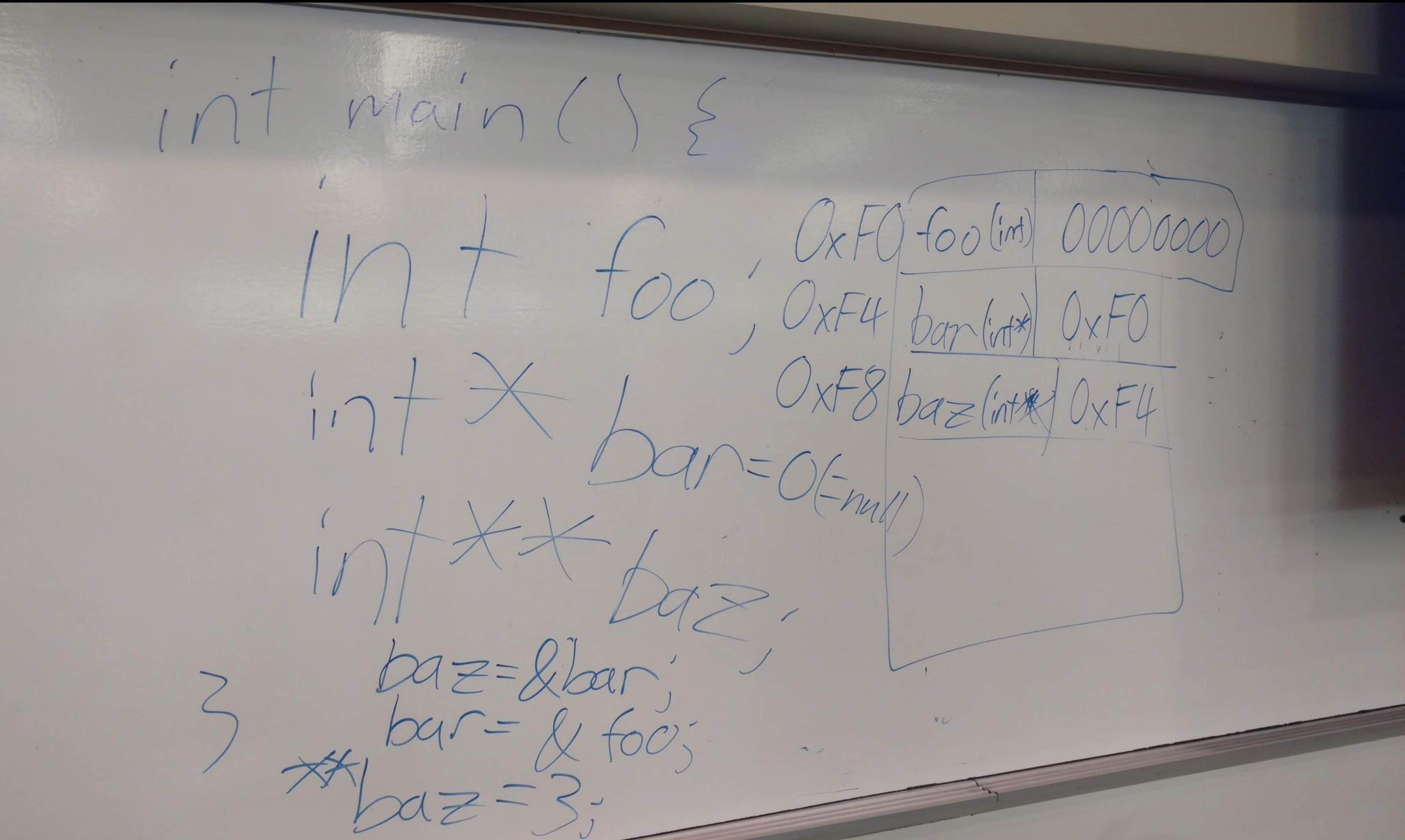
```
// some static variables
int foo;
int *bar;
int **baz;

int main(){
  // set bar to address of foo
  bar = &foo;

  // set baz to address of bar
  baz = &bar;

  // double-dereference baz
  // to alter foo
  **baz = 3;
}
```

Variable Table

- These variables will be in the static section of memory. Let's just call that 0xF0, for example purposes only.

Name	Address	Value
foo	0xF0	3
bar	0xF4	0xF0
baz	0xF8	0xF4

Whiteboard from class:



```
int main() {
    int foo;
    int* bar=0(=null)
    int** baz;
    baz=&bar;
    bar= & foo;
  **baz=3;
}
```

0xF0	foo (int)	00000000
0xF4	bar (int*)	0xF0
0xF8	baz (int**)	0xF4

Pointers and Functions

- Next class we will pass pointers as arguments to functions.
  - This is how you accomplish passing "by reference" in C
    - C does not pass by reference, but Java does for objects.
    - C actually passes by value, it's just that you're providing a pointer AS the value when.
- To be continued...