CSCI 1300

Intro to Computing

Gabe Johnson

Lecture 33 April 8, 2013

Pointers

Lecture Goals

1. Pointers

Upcoming Homework Assignment

HW #8 **Due: Friday, Apr 12**

Pointers, Arrays, Structs

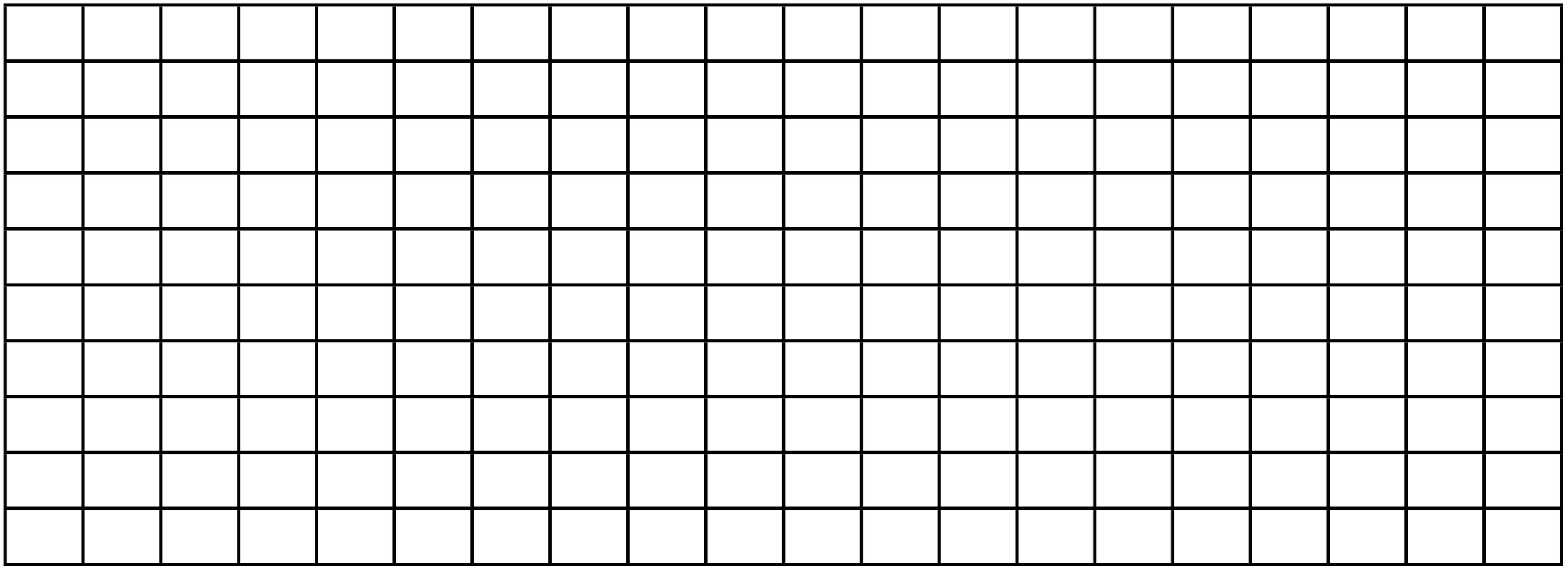
This assignment covers Pointers, C++ Arrays, and C++ Structures. It will give you up to 15 points, but it is only out of 10 points, so you can get 5 points extra credit on this if you get 15/10 before it is due.

Pointers


MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A
0x6339392C,
0x7363682E.
I HATE YOU.

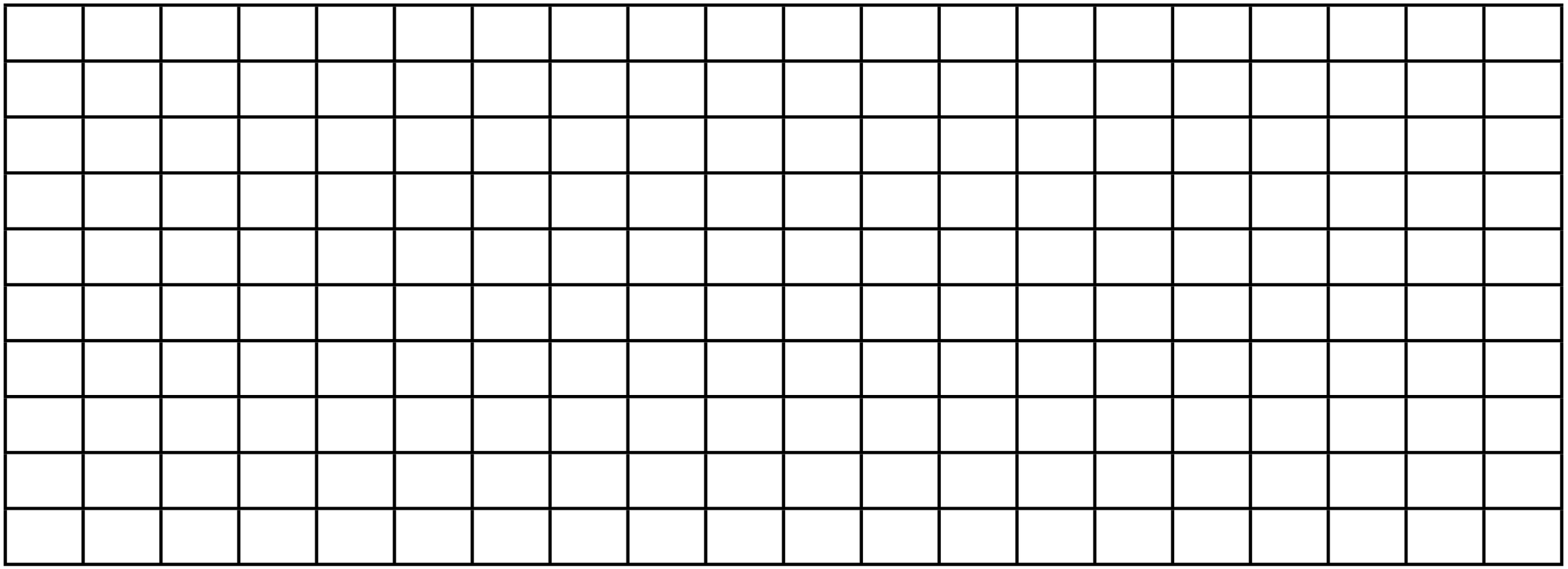
Memory

A CPU has to store data in *memory*. This is often called RAM (random access memory) but it could actually be something else. It is *not* a hard disk, though.



Memory

We can draw memory like this. This has 20 columns and 10 rows, so this memory has 20 * 10 = 200 cells. A real computer has BILLIONS of these cells.



Memory

Lets say we have a memory like below. There's 100 slots, going sequentially from left to right, top to bottom.

	0	1	2	3	4	5	6	7	8	9
0										
10										
20										
30										
40										
50										
60										
70										
80										
90										

Memory

When we create a new variable in C++, the computer has to allocate memory to store it. Let's make an int:

```
int x = 9001;
```

	0	1	2	3	4	5	6	7	8	9
0										
10										
20										
30										
40										
50										
60										
70										
80										
90										

Memory

First the computer find some spot in memory to put the variable. I arbitrarily chose address 37 here. It is highlighted.

	0	1	2	3	4	5	6	7	8	9
0										
10										
20										
30								9001		
40										
50										
60										
70										
80										
90										

Memory

37 is that cell's *address*.

	0	1	2	3	4	5	6	7	8	9
0										
10										
20										
30										
40										
50										
60										
70										
80										
90										

Memory

37 is that cell's *address*. That's variable x's address.
9001 is that cell's *value*. That's variable x's value.

	0	1	2	3	4	5	6	7	8	9
0										
10										
20										
30								9001		
40										
50										
60										
70										
80										
90										

Address Of Operator

An ampersand in front of a variable like this says “I demand you give me the address of this variable!”

Memory

We can prove to ourselves this is true:

```
cout << x << endl; // outputs 9001
cout << &x << endl; // outputs 37
```

	0	1	2	3	4	5	6	7	8	9
0										
10										
20										
30								9001		
40										
50										
60										
70										
80										
90										

Pointer Variables

int* xp = &x;

The address-of operator produces a *pointer.* We write pointers as <data_type>*. E.g.:
int*, float*, string*, bool*, and so on.

Memory

Lets create a pointer variable to x called xp.

```
int* xp = &x;
```

I put it in a cell, since it has to be stored somewhere.

	0	1	2	3	4	5	6	7	8	9
0										
10										
20										
30								9001		
40										
50										
60			37							
70										
80										
90										

Dereference Operator

```
int z = *xp;
```

We can *dereference* a pointer by putting an asterisk in front of it. This gives us the *value* of the variable stored in the memory slot it refers to.

Memory

Lets create a pointer variable to x called xp.

```
int z = *xp;
```

I put it in a cell, since it has to be stored somewhere.

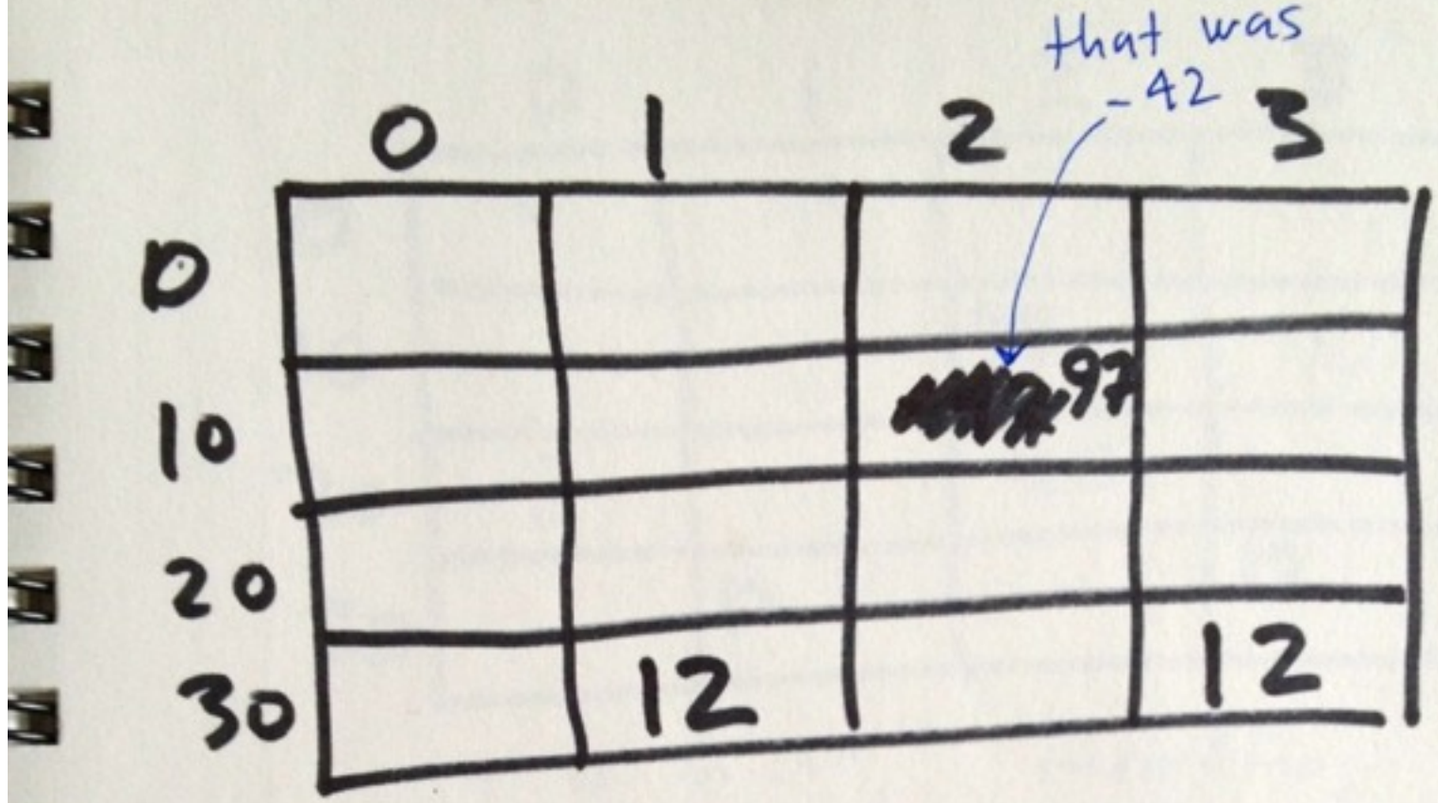
	0	1	2	3	4	5	6	7	8	9
0										
10										
20										
30								9001		
40										
50										
60			37							
70							9001			
80										
90										

Pointers on Paper

Next two slides were done in class.

You can do this sort of thing on your own to practice with the concept.

All the numbers you see are *values*. We can interpret them however we like: as an int, an int*, or zanier things like int***.



Var	add
x	12
y	31
z	33

```
int x = -42;
cout << x << endl;
cout << &x << endl;
int* y = &x;
int *z = &x;
cout << *y << endl; //-42
*y = 97; //change -42 to 97
cout << *z << endl; //97
cout << x << endl; //97
```

```
x = 97;
*y = 97;
*z = 97;
```

	0	1	2	3
0	4			
10			97	0
20	8		20	
30		12		12

var	addr.
x	12
y	31
z	33
k	0
j	20
a	13
b	22

x is an int
&x is an int*
y is an int*
*y is an int
&y is an int**
int** n = &y;
int*** crazy = &n;

foo* i = &foo;

```
int k = 4;
int j = 8;
manhandle(&k, &j);
```

```
void manhandle(int* a, int* b) {
    int before = *a; // 4
    int other = *b; // 8
    *a = other;
    *b = before;
}
```

k will be 8
j will be 4