Pointers

Matthew Mussomele

What Are Variables?

From what we've seen so far, variables are just names that hold some value.

We call out a variable's name and it shouts back its value.

But there's a lot more going on in the background.

Variables are actually just blocks of binary in your computer's memory.

What Are Variables?

You can think of memory as a series of boxes, arranged in a line.

Each box holds one unit (byte) of memory.

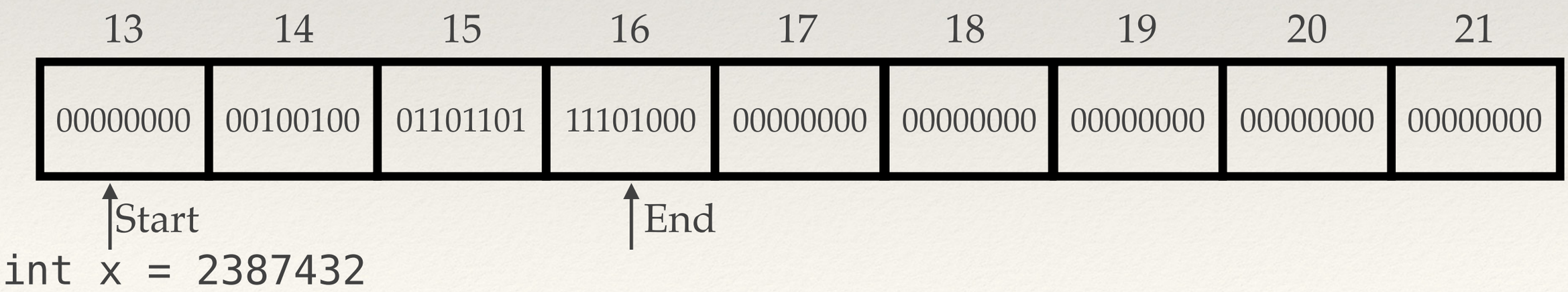
When you use a variable name, the runtime environment actually just figures out what memory location that name refers to and goes there.

Memory Addresses

Memory addresses are sequential, meaning 3 comes before 4 which comes before 5 and so on.

Variables or data larger than a byte take up multiple addresses.

Integers are 32 bits (4 bytes) so they take up 4 boxes.

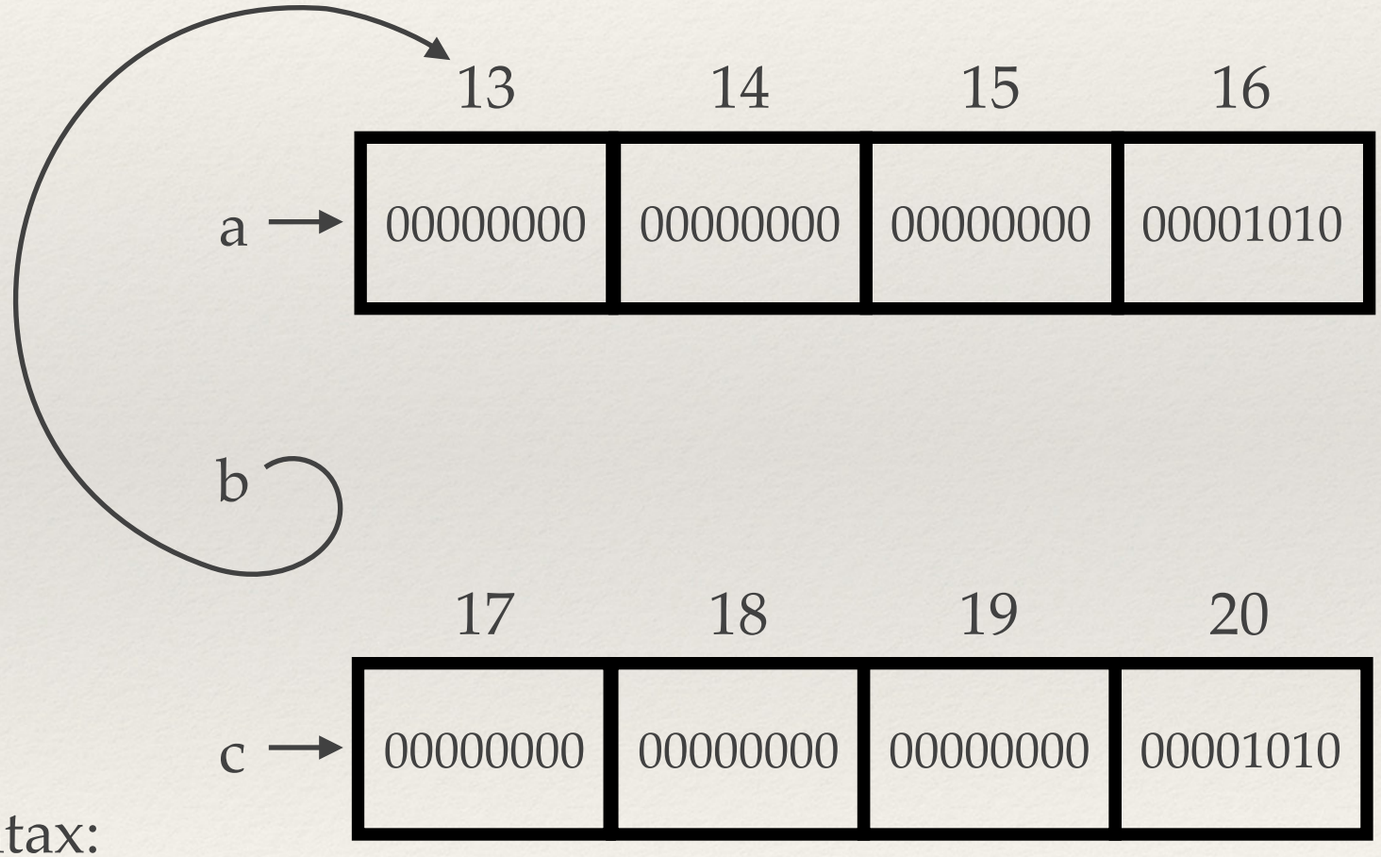


Finding a Variable's Address

You can use the address-of operator (&) to get a variable's location in memory.

```
int a = 10;

int * b = &a;
int c = a;
```



b is what is called a pointer.
It 'points' to a.
Pointers are declared with the syntax:

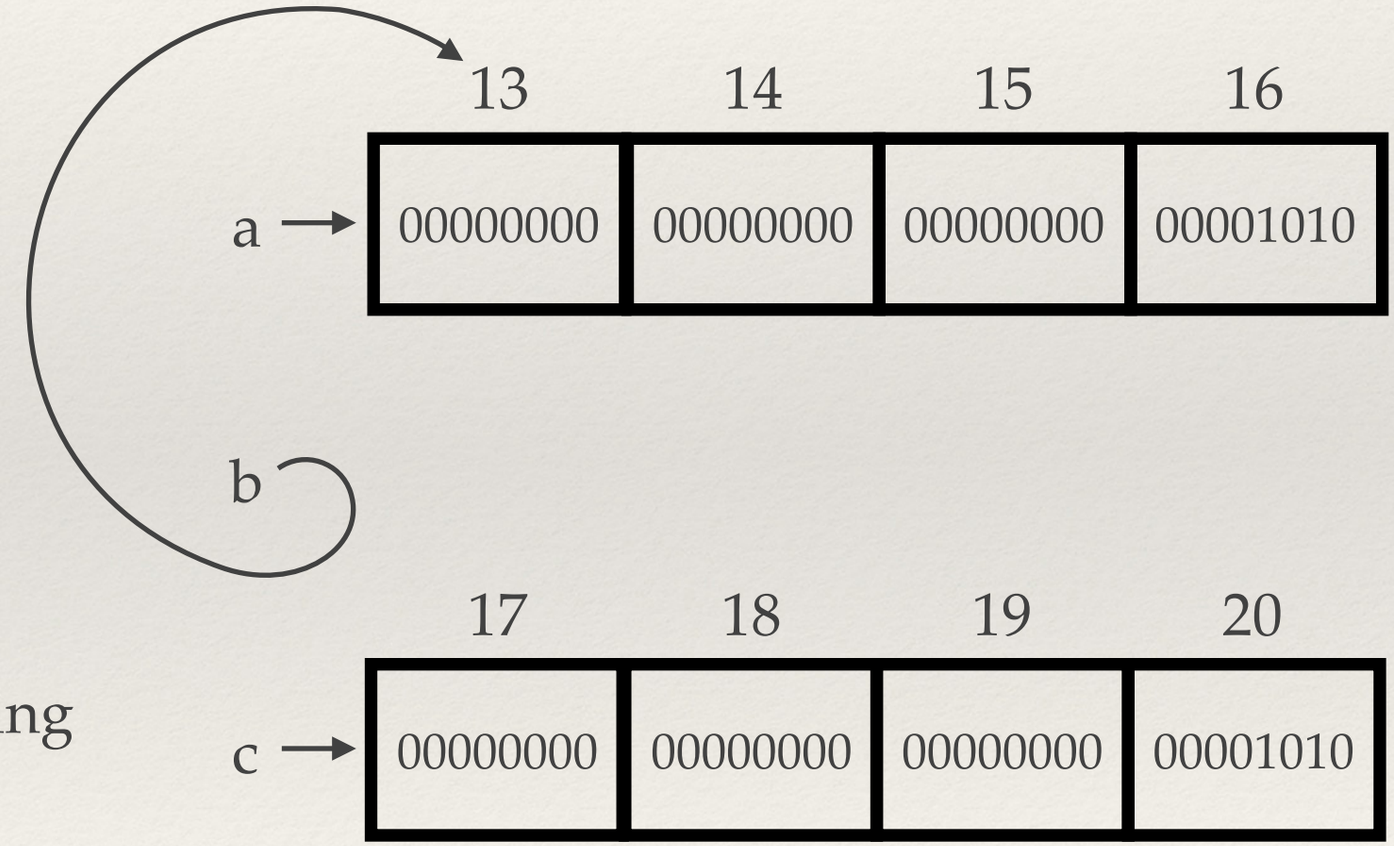
```
<type> * <name>;
<type> * <name> = <value>;
```

Finding a Pointer's Value

To get the value of the variable a pointer refers to, use the dereference operator (*).

```
int a = 10;

int * b = &a;
int c = *b;
```



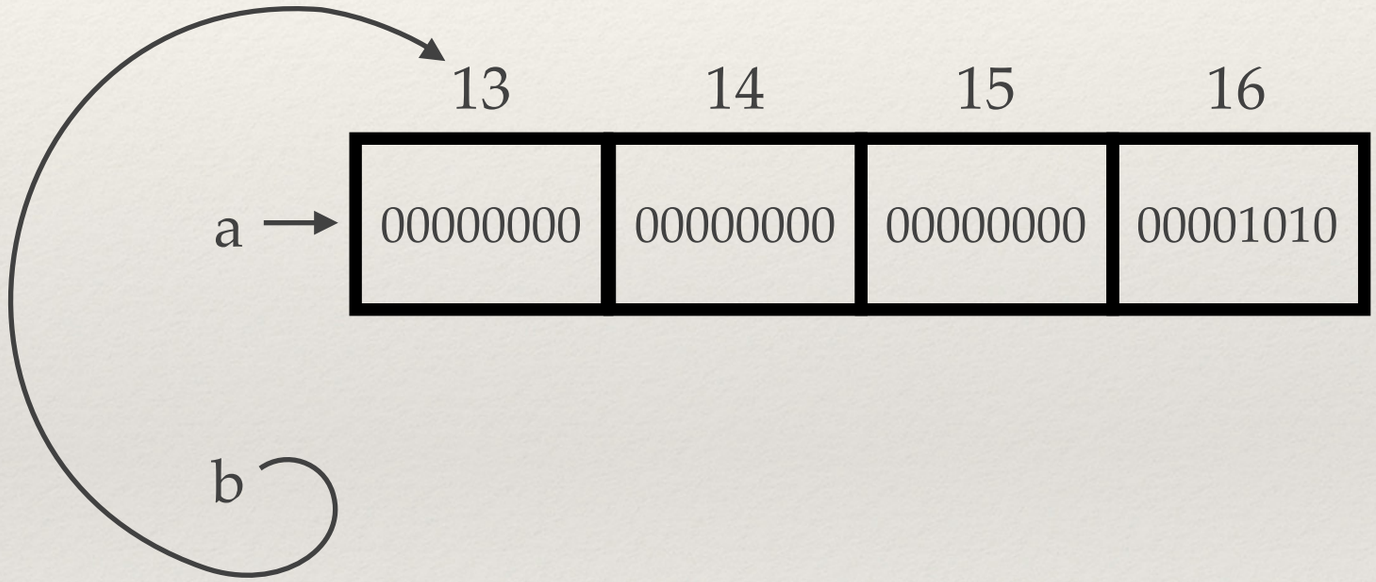
The code above does the same thing as the code on the last slide.

Changing The Value of a Pointer's Target

In order to change the value of the thing a pointer refers to, use the * operator in an assignment statement.

```
int a = 10;

int * b = &a;
*b = 15;
```

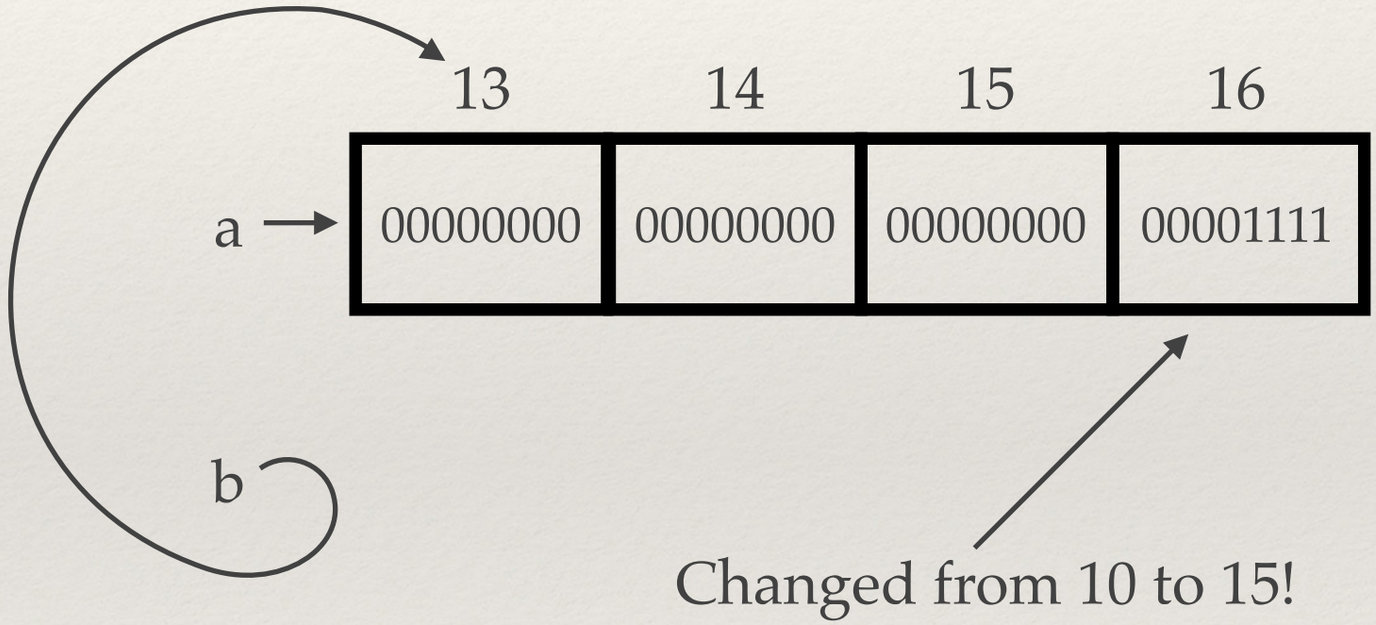


Changing The Value of a Pointer's Target

In order to change the value of the thing a pointer refers to, use the * operator in an assignment statement.

```
int a = 10;

int * b = &a;
*b = 15;
```



The * Symbol

By now you may have noticed that the dereference symbol is exactly the same as the symbol to declare a pointer.

This can be confusing, but try to remember that they are actually *two completely different* symbols.

As a rule of thumb, pointers are on the left of an equals sign when declaring a value and dereferences are most other situations.

Pointers to Pointers

Pointers are variables just like any other.

You can make pointers that point to other pointers.

```
int a = 10;
int * b = &a;
int ** c = &b;

a == *b;  // True
b == *c;  // True
a == **c; // True
```

Pointer Arithmetic

You can do some math with pointers just like you can with regular variables.

This is limited to addition and subtraction.

You can use this to move a pointer to later or earlier variables in memory.

Pointer Arithmetic: Examples

Imagine we execute the lines below:

```
char * a;
int * b;
double * c;
// assume they all equal 0
a++;
b++;
c++;
```

What do you think the values of a, b, and c are?

a = 1
b = 2
c = 8

Note: The code shown won't run properly.

Pointer Arithmetic

You can't do direct arithmetic with pointers.

Otherwise you could make one point to the middle of a variable instead of the start.

Instead of thinking of incrementing a pointer as increasing it's value by one, think of it as increasing it's value by one 'memory unit'.

The size of a 'memory unit' is the number of bytes of the type the pointer refers to.

Read-only Pointers

We've seen that pointers can both get the value of what it points to and actually change it.

You can declare pointers so that they can only get the value and not change it using the `const` keyword.

```
int a = 10;
const int * b = &a;
cout << *b << endl; // will print 10
*b = 15;            // will error, b is 'read only'
```

This makes b a constant, meaning that it cannot be changed.

Read-only Pointer Example

Read-only pointers can be useful when you have a function that takes in pointers, but you don't want it to be able to change the value of the thing the pointer refers to.

You can do this by making the formal parameters constants.

```
int add_one(const int * a) {
    return *a + 1;
}
```

This will compile, we can access the value of a read-only pointer.

This won't compile, we can't change the value of what a read-only pointer refers to.

```
int add_one(const int * a) {
    *a += 1;
    return *a;
}
```

String Pointers

String literals in C++ are just arrays of characters.

We will talk more about arrays later in the course.

You can use pointer arithmetic to access characters of a string.

```
char * a = "This is a string literal.";
cout << *(a + 1);   // this will print 'h'
```

Null Pointers

Null pointers are just what they sound like, they point to nothing.

You declare a null pointer using the following syntax:

```
int * a = 0; // null pointer set by making it equal zero
```

Function Pointers

Function pointers are pointers that reference functions.

You can use this to pass function references to other functions, and you can access the function a pointer refers to just like normal pointers.

The syntax is as follows:

```
<return type> (* <name>)(<parameter type>);
```

Citations

- "Pointers." C++ Tutorials. Cplusplus.com, n.d. Web. 12 June 2015. <http://www.cplusplus.com/doc/tutorial/pointers/>. Used as documentation-style reference.