CS 161, Lecture 15: Pointers and Memory Model

Warm-Up

Function: increments_by

Description: increments num1 by the value num2

Input: num1, num2

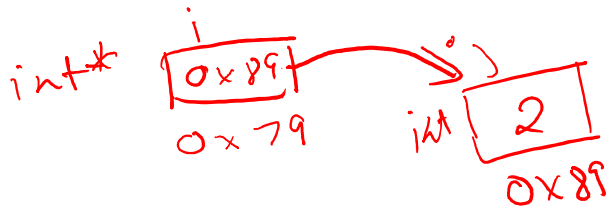
Return type: void

void increments_by (int num1, int num2)

num1 = num1 + num2

	Pass By Value	Pass By Reference	Pass By Pointer
Parameter Listing	int num1, int num2	int &num1, int &num2	int *num1, int *num2
What is actually being passed?	copy of a value	value and memory address	memory address
Does the function body need to change? If so, how?	No	No	Yes Add * dereference operator to alter values
Function Call	increments_by(num1, num2) →		increments_by(&num1, &num2)
How will things change from where the function was called?	They stay the same	They change num1 = num1 + num2	→

More Pointers



- Declaring pointers

```
int *i; //This will hold an int memory address
int j; //This holds an int
i = &j; //set the address that i holds to the address of j
j = 2; //sets the value of j to 2, *i is 2
(*i)++; //increments the value i points to, j is 3
```

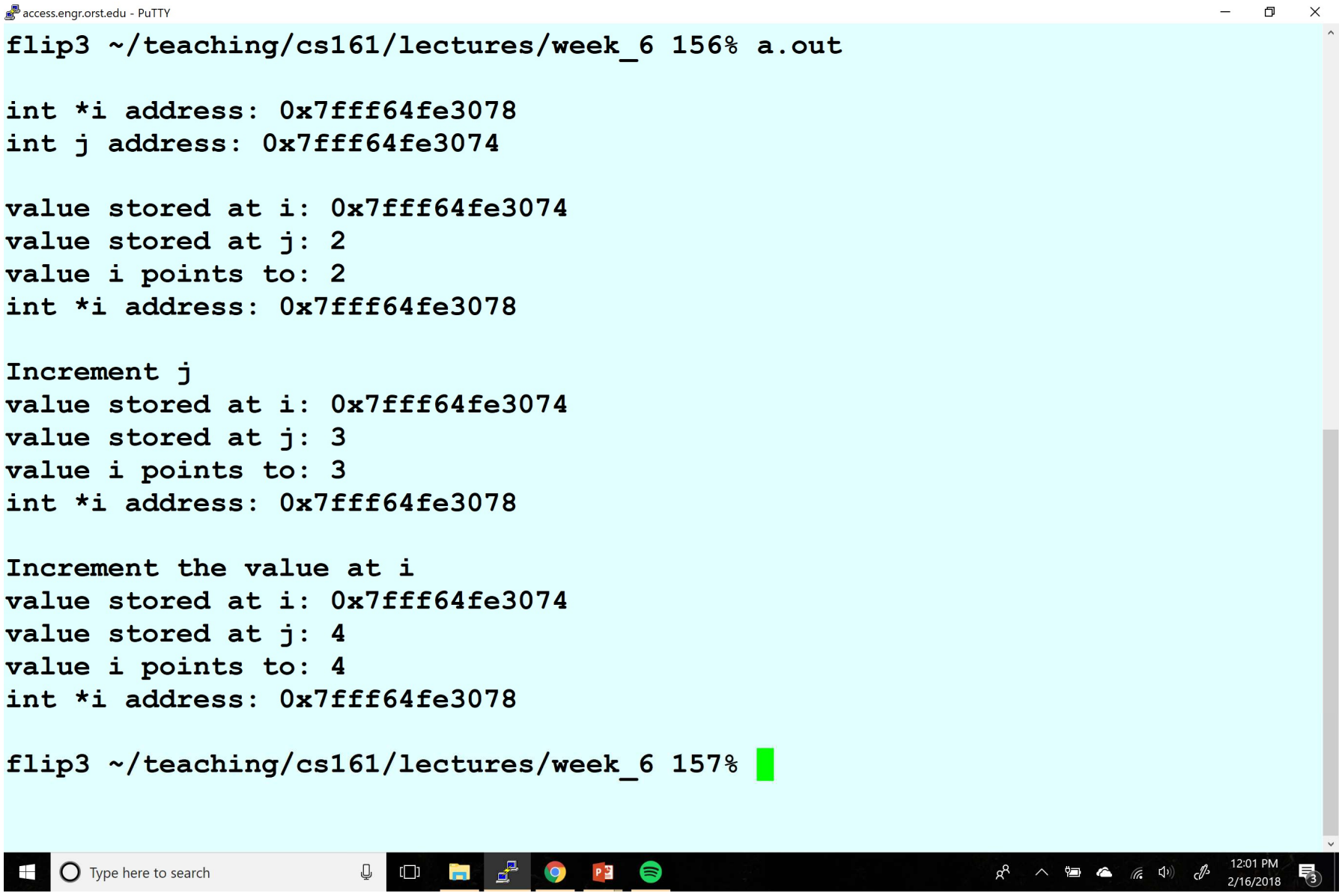
Demo

access.engr.orst.edu - PuTTY

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 int main() {
 6
 7         int *i;
 8         int j;
 9         cout << endl;
10         cout << "int *i address: " << &i << endl;
11         cout << "int j address: " << &j << endl;
12
13         i = &j;
14         j = 2;
15         cout << endl;
16         cout << "value stored at i: " << i << endl;
17         cout << "value stored at j: " << j << endl;
18
19         cout << "value i points to: " << *i << endl;
20         cout << "int *i address: " << &i << endl;
21         j++;
22         cout << endl;
23         cout << "Increment j" << endl;
24         cout << "value stored at i: " << i << endl;
"pointers.cpp" 41L, 903C                                  24,2-9        Top
```

access.engr.orst.edu - PuTTY

```
19          cout << "value i points to: " << *i << endl;
20          cout << "int *i address: " << &i << endl;
21          j++;
22          cout << endl;
23          cout << "Increment j" << endl;
24          cout << "value stored at i: " << i << endl;
25          cout << "value stored at j: " << j << endl;
26
27          cout << "value i points to: " << *i << endl;
28          cout << "int *i address: " << &i << endl;
29          cout << endl;
30
31          (*i)++;
32          cout << "Increment the value at i" << endl;
33          cout << "value stored at i: " << i << endl;
34          cout << "value stored at j: " << j << endl;
35
36          cout << "value i points to: " << *i << endl;
37          cout << "int *i address: " << &i << endl;
38          cout << endl;
39
40          return 0;
41  }
~
                                                    24,2-9          Bot
```

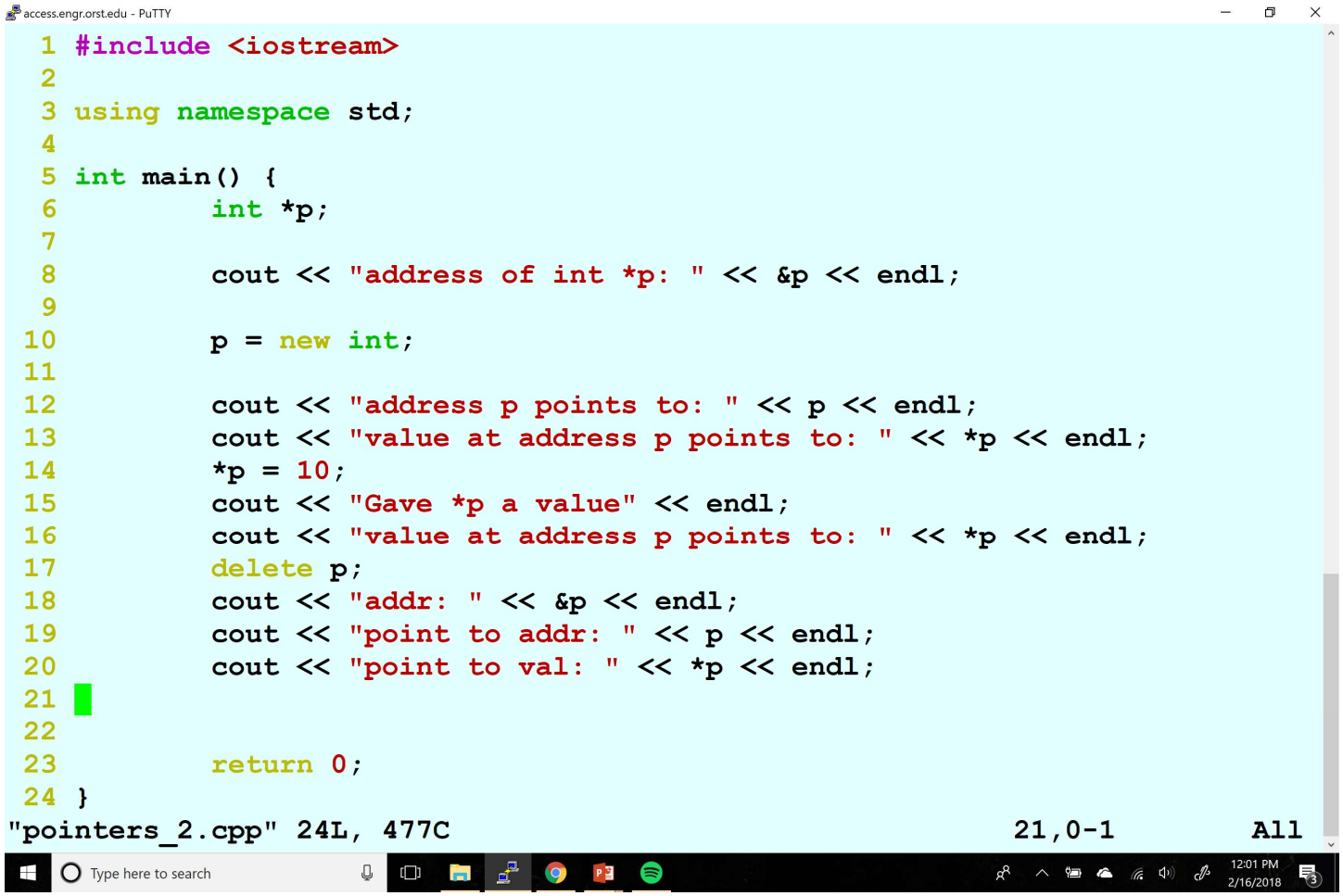

access.engr.orst.edu - PuTTY
flip3 ~/teaching/cs161/lectures/week_6 156% a.out
int *i address: 0x7fff64fe3078
int j address: 0x7fff64fe3074
value stored at i: 0x7fff64fe3074
value stored at j: 2
value i points to: 2
int *i address: 0x7fff64fe3078
Increment j
value stored at i: 0x7fff64fe3074
value stored at j: 3
value i points to: 3
int *i address: 0x7fff64fe3078
Increment the value at i
value stored at i: 0x7fff64fe3074
value stored at j: 4
value i points to: 4
int *i address: 0x7fff64fe3078
flip3 ~/teaching/cs161/lectures/week_6 157%
Type here to search
12:01 PM
2/16/2018

What if we don't have an address to point to?

- We make one with the **new** keyword (dynamically allocate)

```
int *p;
p = new int; //new returns an address
*p = 10;
```

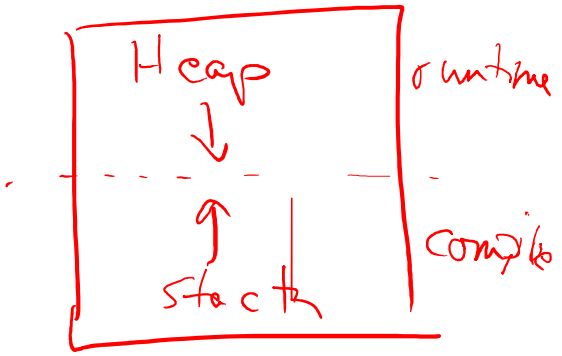
Demo


access.engr.orst.edu - PuTTY
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6 int *p;
7
8 cout << "address of int *p: " << &p << endl;
9
10 p = new int;
11
12 cout << "address p points to: " << p << endl;
13 cout << "value at address p points to: " << *p << endl;
14 *p = 10;
15 cout << "Gave *p a value" << endl;
16 cout << "value at address p points to: " << *p << endl;
17 delete p;
18 cout << "addr: " << &p << endl;
19 cout << "point to addr: " << p << endl;
20 cout << "point to val: " << *p << endl;
21
22
23 return 0;
24 }
"pointers_2.cpp" 24L, 477C
21,0-1
All
Type here to search
12:01 PM
2/16/2018

Different Types of Memory

- CPU: central processing unit, “brain” of the computer system
- Main memory
  - where current programs are executed
  - CPU has direct and quick address
  - Volatile: contents are lost when the power goes out
- Secondary Memory
  - Nonvolatile, long term storage
  - Ex. Files, hard drive, USB, etc.

How Main Memory is Structured



- Stack
  - Variables defined at compile time go on the stack (global variables, constants)
  - Functions have their own stack frame
  - When a function ends, the stack frame collapses and cleans up the memory for you -> sometimes referred to as automatic variables
- Heap
  - Variables defined at runtime (**new** keyword)
  - Variables declared dynamically in a function do not disappear when the function ends as they are on the heap and not the function stack
  - Can run out of heap space
  - Need to free dynamic memory when done with it, otherwise memory leaks

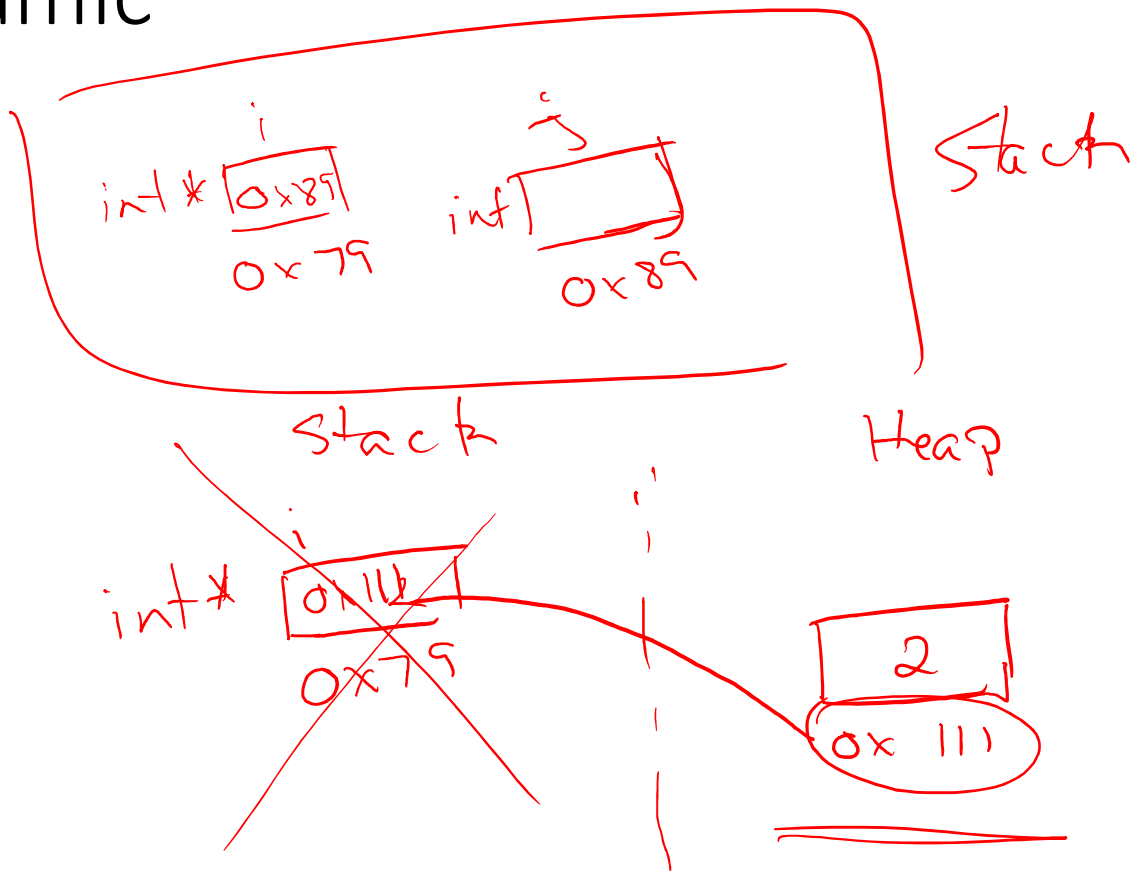
Static vs Dynamic

- Static

```
int *i, j=2;
i = &j;
```

- Dynamic

```
int *i = NULL;
i = new int;
*i = 2;
```

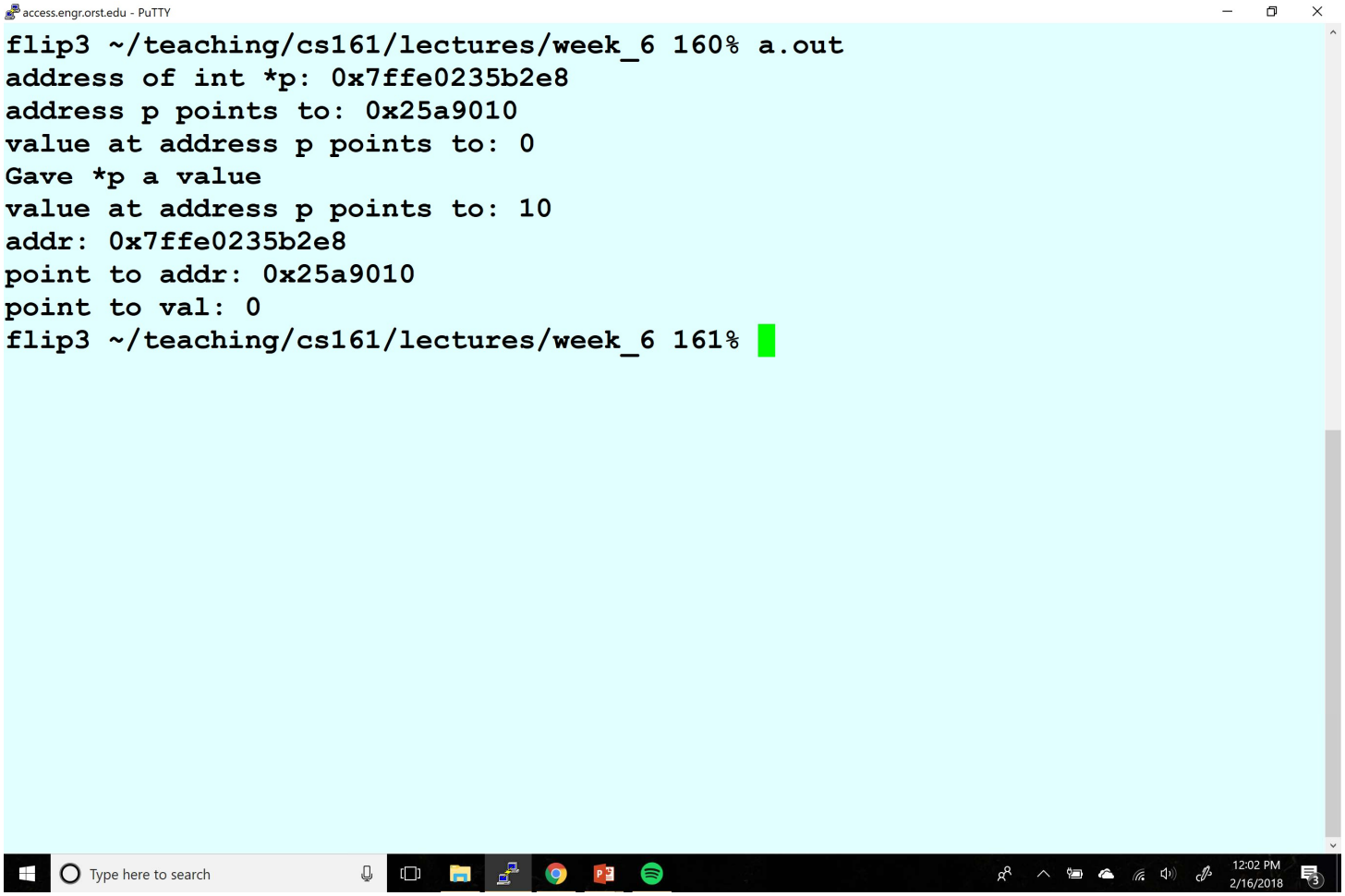


Fixing Memory Leaks

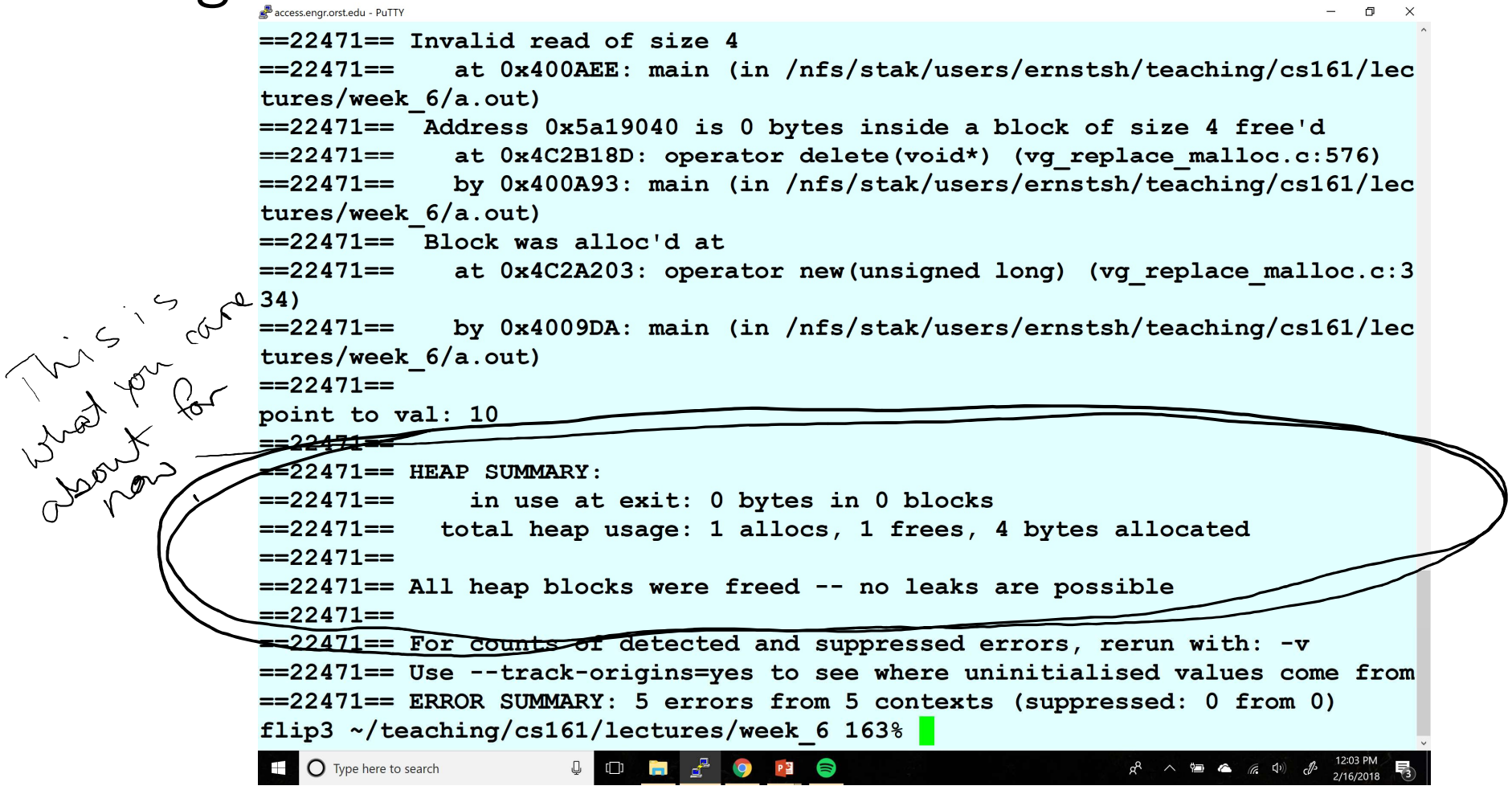
- How to tell you may have a memory leak
  - You used the **new** keyword
  - You never used the **delete** keyword
  - You run valgrind
    - Compile and produce an executable for your program
    - Run valgrind with your executable (valgrind executable_name)
- How to fix memory leaks
  - Delete dynamic memory when you are done with it

```
int *i = new int;
delete i;
```

Demo


access.engr.orst.edu - PuTTY
flip3 ~/teaching/cs161/lectures/week_6 160% a.out
address of int *p: 0x7ffe0235b2e8
address p points to: 0x25a9010
value at address p points to: 0
Gave *p a value
value at address p points to: 10
addr: 0x7ffe0235b2e8
point to addr: 0x25a9010
point to val: 0
flip3 ~/teaching/cs161/lectures/week_6 161%
Type here to search
12:02 PM
2/16/2018

valgrind



This is what you care about for now

Feedback

https://tinyurl.com/y7c79hap