Review for Test

October 11, 2017

(beginning at 6:10pm)

Info

http://docs.cs50.net/2017/fall/test/about.html

- 72 hour window in which to take the test.
  - You should require much less than that. Expect to spend an average of 30 minutes per question.

- Released Fri 10/13 at noon, due via submit50 Mon 10/16 at noon.
  - Be sure to run `update50` in your IDE before submitting!
  - Submitting seven minutes late is equivalent to not submitting at all; <u>don’t</u> wait until the last possible second.

Resources

- Consult the syllabus for a guide of topics.
    - We’ll run through everything at a very high level today.
- Review lecture notes.
- Review lecture source code.
- Review lecture slides.
- (Re)watch lecture videos and shorts.
- Review last year’s test and answer key.
- Review problem set specifications, distribution code, and sample solutions.

Resources

- Office hours
    - Tonight in Widener from 8–10pm.
    - Tomorrow at HSA from 10am–5pm.
    - Tomorrow night in Widener from 8–10pm.

Resources

- Office hours
  - Tonight in Widener from 8–10pm.
  - Tomorrow at HSA from 10am–5pm.
  - Tomorrow night in Widener from 8–10pm.

- No office hours during the Test (10/13 through 10/16) up through shortly after pset6 is released.
- Office hours resume on Sun 10/22.

Resources

- CS50 Discourse
  - You may post questions through noon on 10/13, and staff will try to answer.
  - You may <u>not</u> post questions on Discourse from Fri 10/13 noon through Mon 10/16.
    - Discourse will become read-only during the Test.

Resources

- CS50 Discourse
  - You may post questions through noon on 10/13, and staff will try to answer.
  - You may not post questions on Discourse from Fri 10/13 noon through Mon 10/16.
    - Discourse will become read-only during the Test.

- The **only** humans to which you may turn for help during the Test are the course's heads.

Resources

- The **only** humans to which you may turn for help during the Test are the course's heads.

Resources

- The **only** humans to which you may turn for help during the Test are the course's heads.

- You may not email any other CS50 staff members.
- You may not ask roommates, friends, tutors, or classmates for help.
- You may not *post* questions on any online forum, whether local to the course or not (though you may review previously-asked questions).

Topics

- Weeks 0–5 (a/k/a Lectures 0–6)
  - Does not cover Thu 10/12 lecture at Yale. (Dynamic Programming)
  - Does not cover Fri 10/13 lecture at Harvard. (Python)
- Problem Sets 0–5.
  - Does not presuppose completion of any “more comfortable” versions of problems.

Topics

- Weeks 0–5 (a/k/a Lectures 0–6)
    - Does not cover Thu 10/12 lecture at Yale. (Dynamic Programming)
    - Does not cover Fri 10/13 lecture at Harvard. (Python)
- Problem Sets 0–5.
    - Does not presuppose completion of any “more comfortable” versions of problems.

- Test will contain some coding exercises, but not on the scale of any of the more recent problem sets.

Week 0

- Binary
  - Digits: 0, 1
  - Place values: 1s, 2s, 4s, 8s, 16s...

Week 0

- Binary
  - Digits: 0, 1
  - Place values: 1s, 2s, 4s, 8s, 16s…
- ASCII
  - Uniform standard for mapping of numbers to characters.
  - ‘A’ is 65, ‘a’ is 97…

Week 0

- Binary
  - Digits: 0, 1
  - Place values: 1s, 2s, 4s, 8s, 16s…
- ASCII
  - Uniform standard for mapping of numbers to characters.
  - ‘A’ is 65, ‘a’ is 97…
- Bytes
  - The value of a byte is context-dependent.
  - Maybe that 65 is just a 65.
  - In Microsoft Word that 65 might indeed be an ‘A’.
  - In Photoshop that 65 might represent the red value of a particular pixel.

Week 0

- Algorithms
    - Step by step sets of instructions for completing a task.
    - Peanut butter and jelly.
        - Anticipating errors, and the importance of precision.
    - Finding Mike Smith in a phone book.
    - Correctness versus efficiency.

Week 0

- Algorithms
  - Step by step sets of instructions for completing a task.
  - Peanut butter and jelly.
    - Anticipating errors, and the importance of precision.
  - Finding Mike Smith in a phone book.
  - Correctness versus efficiency.

- Pseudocode
  - English-like syntax that can be used as a stepping stone to solving a problem.
  - Functions, statements, Boolean expressions, loops…

Week 0

- Scratch
  - Basic blocks – control, data, sound, looks.
  - Custom blocks – “functions”.
  - Events – when _______________.

Week 1

- Loops
    - `for` – running a specific number of times.
    - `while` – running some number of times, possibly zero.
    - `do-while` – running some number of times, at least once.

Week 1

- Loops
  - `for` – running a specific number of times.
  - `while` – running some number of times, possibly zero.
  - `do-while` – running some number of times, at least once.
- Conditions
  - Boolean expressions – true or false
  - `if`, `else if`, `else`
  - `switch`
  - Ternary operator - `?:`

Week 1

- Variables
  - Containers that hold information.
  - Before using, need to declare.
  - Variables hold information of a specific type, and have a name.
  - Use = to assign values to variables, right-to-left.

Week 1

- Variables
  - Containers that hold information.
  - Before using, need to declare.
  - Variables hold information of a specific type, and have a name.
  - Use `=` to assign values to variables, right-to-left.
- Compiling
  - `make` is a utility we use to turn our C code into executable programs.
  - `clang` is a compiler that does the hard work of this translation.
  - Computers only understand machine code, not our C source.
  - Preprocessing → compiling → assembling → linking.

Week 1

- Data Types
  - Native data types in C
    - **int**, **char**, **float**, **double**, **long**
  - Additional data types
    - **bool** (via **stdbool.h**, itself included in **cs50.h**)
    - **string** (via **cs50.h**)
  - **signed** and **unsigned**
  - 1 byte
    - **bool**, **char**
  - 4 bytes
    - **float**, **int**
  - 8 bytes
    - **double**, **long**, **string**

Week 1

- Functions
  - Functions are **abstractions** that allow us to “outsource” aspects of our problem.
  - Black box model.
  - Prototypes versus definitions.
  - Prototypes versus function calls.
  - Return types and parameters.

Week 1

```
int square(int n);

int main(void)
{
    int x = get_int("Integer please: ");
    int squared = square(x);
    printf("%i squared is %i.\n", x, squared);
}

int square(int n)
{
    return n * n;
}
```

Week 1

```
int square(int n);

int main(void)
{
    int x = get_int("Integer please: ");
    int squared = square(x);
    printf("%i squared is %i.\n", x, squared);
}

int square(int n)
{
    return n * n;
}
```

Week 1

```
int square(int n);

int main(void)
{
    int x = get_int("Integer please: ");
    int squared = square(x);
    printf("%i squared is %i.\n", x, squared);
}

int square(int n)
{
    return n * n;
}
```

Week 1

```
int square(int n);

int main(void)
{
    int x = get_int("Integer please: ");
    int squared = square(x);
    printf("%i squared is %i.\n", x, squared);
}

int square(int n)
{
    return n * n;
}
```

Week 1

```
int square(int n);

int main(void)
{
    int x = get_int("Integer please: ");
    int squared = square(x);
    printf("%i squared is %i.\n", x, squared);
}

int square(int n)
{
    return n * n;
}
```

Week 1

```
int square(int n);

int main(void)
{
    int x = get_int("Integer please: ");
    int squared = square(x);
    printf("%i squared is %i.\n", x, squared);
}

int square(int n)
{
    return n * n;
}
```

Week 1

```
int square(int n);

int main(void)
{
    int x = get_int("Integer please: ");
    int squared = square(x);
    printf("%i squared is %i.\n", x, squared);
}

int square(int n)
{
    return pow(n, 2);
}
```

Week 1

```
int square(int n);

int main(void)
{
    int x = get_int("Integer please: ");
    int squared = square(x);
    printf("%i squared is %i.\n", x, squared);
}

int square(int n)
{
    int product = 0;
    for (int i = 0; i < n; i++)
    {
        product += n;
    }
    return product;
}
```

Week 1

- Overflow
    - With an integer, we only have 4 bytes (32 bits) to work with.
    - We can’t store any number equal to or greater than 2^{32}.

Week 1

- Overflow
  - With an integer, we only have 4 bytes (32 bits) to work with.
  - We can’t store any number equal to or greater than 2^{32}.
- Imprecision
  - With a float, we only have 4 bytes (32 bits) to work with.
  - We cannot possibly represent every real number.

Week 1, continued

- Bugs and Tools
  - Implicit declaration of functions.
  - Use of undeclared identifier.
  - Out of bounds error.
  - Segmentation fault.
  - `help50`, `debug50`, `check50`, `style50`.
  - Breakpoints, step over, step into.
  - `eprintf`.
  - Later in the course: `valgrind`.

Week 1, continued

- Reference Tools
  - Manual pages are part of most Linux installations.
    - `man` __________.
  - `reference.cs50.net` is written by the staff.
  - Many online equivalents for C and other languages.

Week 1, continued

- Reference Tools
  - Manual pages are part of most Linux installations.
    - `man` ____________.
  - `reference.cs50.net` is written by the staff.
  - Many online equivalents for C and other languages.

- Cryptography
  - Art and science of obscuring information.
  - Rotational cipher.

Week 1, continued

- Strings
  - A sequence of characters.

Week 1, continued

- Strings
  - ~~A sequence of characters.~~
  - An array of characters.
  - Length of a string is available via the function **`strlen`**.
  - Each character of the string is available with **`str[i]`**
    - **`0 <= i < strlen(str)`**
  - All strings end with the **`\0`** character.

Week 1, continued

- Strings
  - ~~A sequence of characters.~~
  - An array of characters.
  - Length of a string is available via the function `strlen`.
  - Each character of the string is available with `str[i]`
    - `0 <= i < strlen(str)`
  - All strings end with the `\0` character.
- Typecasting
  - Think back to ASCII, every character is associated with a number.
  - We can treat characters as numbers and do math with them using their ASCII values.
  - Explicit typecasting uses a `(type)` specifier.

Week 1, continued

- Command-Line Arguments
  - By modifying our prototype for **`main`**, the user can supply extra information to our programs at runtime.
  - **`int main(int argc, string argv[])`**
  - **`argc`** refers to how many things the user typed.
  - **`argv`** is an array of strings storing what they actually typed.

Week 2

- Searching
  - Linear search considers a general array, and looks over each element from beginning to end until it finds the target.
  - Binary search considers a **sorted** array, looks at the middle, and discards half of the remaining array until it finds the target.

Week 2

- Searching
  - Linear search considers a general array, and looks over each element from beginning to end until it finds the target.
  - Binary search considers a **sorted** array, looks at the middle, and discards half of the remaining array until it finds the target.
- Sorting
  - Selection sort: Find the smallest remaining, swap with the first.
  - Bubble sort: Adjacent pairs out of order? Swap them.
  - Insertion sort: Shift previously sorted elements to make room.
  - Merge sort: Sort partial arrays, then combine them together.

Week 2

- Big O
  - Provides us with a shorthand way to refer to the running time of various algorithms.
  - In CS50, normally O describes the *upper bound* on runtime.
  - In CS50, normally Ω describes the *lower bound* on runtime.

Week 2

Algorithm	Upper bound (O)	Lower bound (Ω)
Linear search	n	1
Binary search	$\log n$	1
Selection sort	n^2	n^2
Bubble sort	n^2	n
Insertion sort	n^2	n
Merge sort	$n \log n$	$n \log n$

Week 2

- Recursion
  - Problem solving technique where we use the solution to a smaller problem to inform the solution to a larger one.
  - Series summation, factorial, exponentiation, Fibonacci sequence…
  - A recursive algorithm has two parts:
    - Base case – recursion stops; the simple case we have a solution for.
    - Recursive case – recursion continues; make a more complex case a little bit simpler, tending towards the base case.

Week 2

```
int fact(int n)
{
    if (n <= 0)
    {
        return 1;
    }
    return n * fact(n-1);
}
```

```
int fact(int n)
{
    int product = 1;
    while (n > 0)
    {
        product *= n--;
    }
    return product;
}
```

Week 2

```
int fact(int n)
{
    if (n <= 0)
    {
        return 1;
    }
    return n * fact(n-1);
}
```

```
int fact(int n)
{
    int product = 1;
    while (n > 0)
    {
        product *= n--;
    }
    return product;
}
```

Week 2

```
int fact(int n)
{
    if (n <= 0)
    {
        return 1;
    }
    return n * fact(n-1);
}
```

```
int fact(int n)
{
    int product = 1;
    while (n > 0)
    {
        product *= n--;
    }
    return product;
}
```

Week 3

- Call Stack
  - Swapping values in a separate function has no effect in the *calling* function.
  - Passing variables to a function gives that function its own local copy of those variables; our original ones remain intact.
  - A function call creates a stack frame.
  - The most recently called function is the one with the “highest” frame on the stack, and is the only function active.
  - All other functions are “on pause” where they left off.

Week 3

- Pointers
  - How can we access memory in other functions?
  - Pointers are addresses, specifically the addresses of variables we care about.
  - Finding a variable's address: &
  - Going to an address to manipulate a variable: *
    - Dereferencing

Week 3

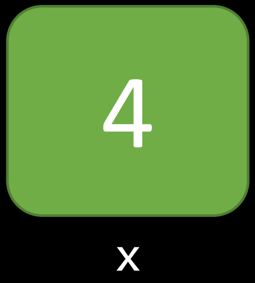
- Pointers
  - How can we access memory in other functions?
  - **Pointers are addresses**, specifically the addresses of variables we care about.
  - Finding a variable’s address: **&**
  - Going to an address to manipulate a variable: *****
    - Dereferencing

Week 3

```
int main(void)
{
    int x = 4;
    int *px = &x;
    *px = 5;
    printf("%i\n", x);
}
```

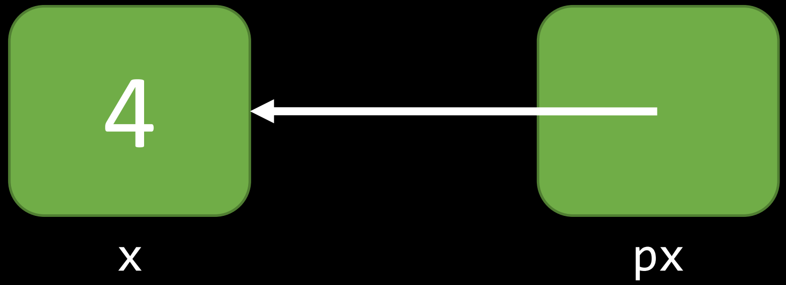
Week 3

```
int main(void)
{
    int x = 4;
    int *px = &x;
    *px = 5;
    printf("%i\n", x);
}
```



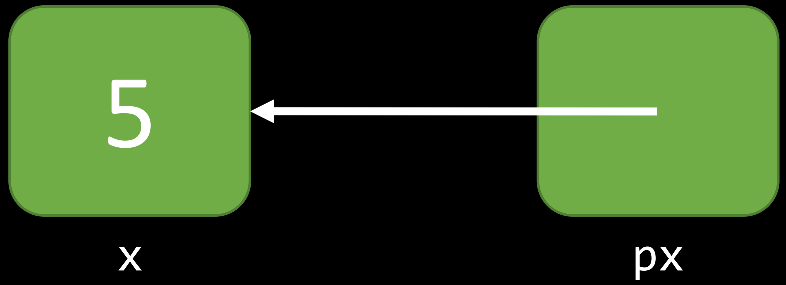
Week 3

```
int main(void)
{
    int x = 4;
    int *px = &x;
    *px = 5;
    printf("%i\n", x);
}
```



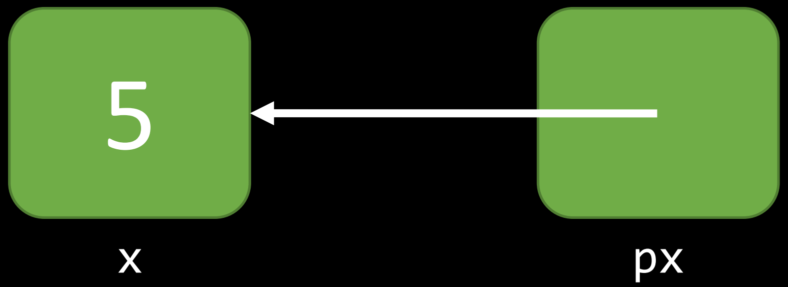
Week 3

```
int main(void)
{
    int x = 4;
    int *px = &x;
    *px = 5;
    printf("%i\n", x);
}
```



Week 3

```c
int main(void)
{
    int x = 4;
    int *px = &x;
    *px = 5;
    printf("%i\n", x);
}
```



```
5
```

Week 3

```
int main(void)
{
    int x = 4;
    change(x);
    printf("%i\n", x);
}
```

```
void change(int x)
{
    x = 5;
    return;
}
```

Week 3

```
int main(void)
{
    int x = 4;
    change(&x);
    printf("%i\n", x);
}
```

```
void change(int *x)
{
    *x = 5;
    return;
}
```

Week 3

- Strings Redux
  - The variable name of a string is behind the scenes just a pointer to (aka the address of) its first character.
  - `string s = "CS50";`
  - `string t = "CS50";`

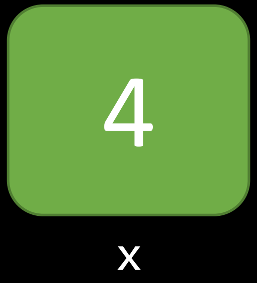
Week 3

- Strings Redux
  - The variable name of a string is behind the scenes just a pointer to (aka the address of) its first character.
  - `string s = "CS50";`
  - `string t = "CS50";`
- Dynamic Memory
  - If I need memory while my program is running that I didn’t anticipate at compile-time, I can use `malloc`.
  - `malloc` expects a number of bytes as a parameter, and gives you back a pointer.
    - `sizeof` is helpful here!
  - Need to `free` all dynamically allocated memory.

Week 3

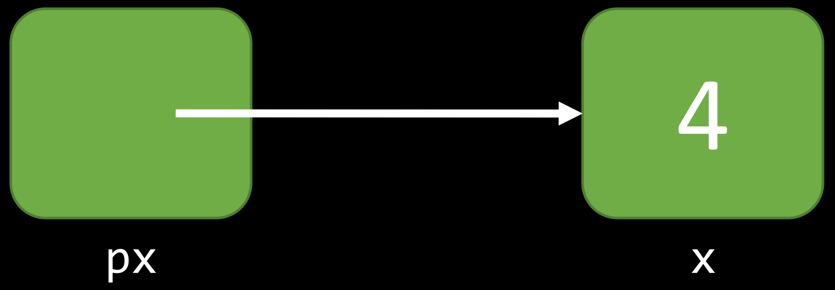
```
int main(void)
{
    int x = 4;
    int *px = &x;
    int *py = malloc(sizeof(int));
    *py = 5;
}
```

Week 3



```
int main(void)
{
    int x = 4;
    int *px = &x;
    int *py = malloc(sizeof(int));
    *py = 5;
}
```

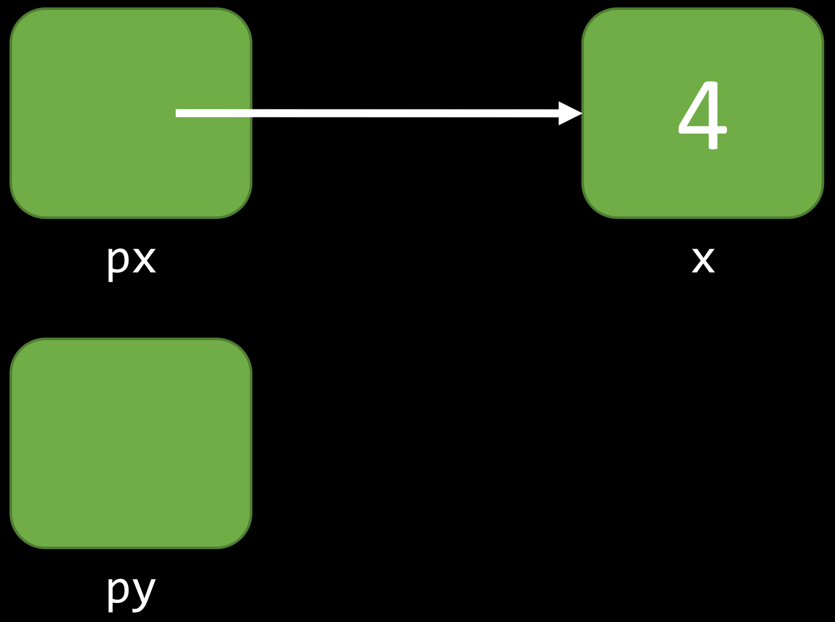
Week 3



```
int main(void)
{
    int x = 4;
    int *px = &x;
    int *py = malloc(sizeof(int));
    *py = 5;
}
```

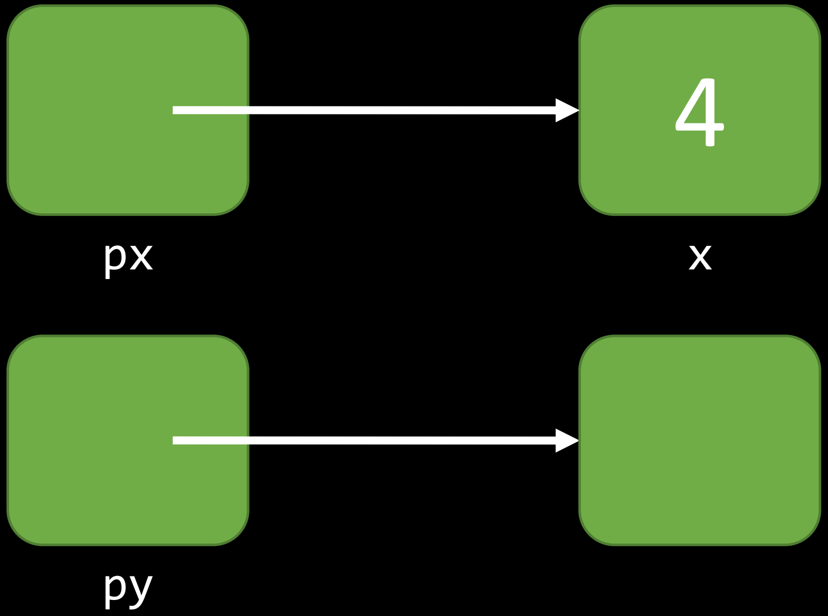
Week 3

```
int main(void)
{
    int x = 4;
    int *px = &x;
    int *py = malloc(sizeof(int));
    *py = 5;
}
```



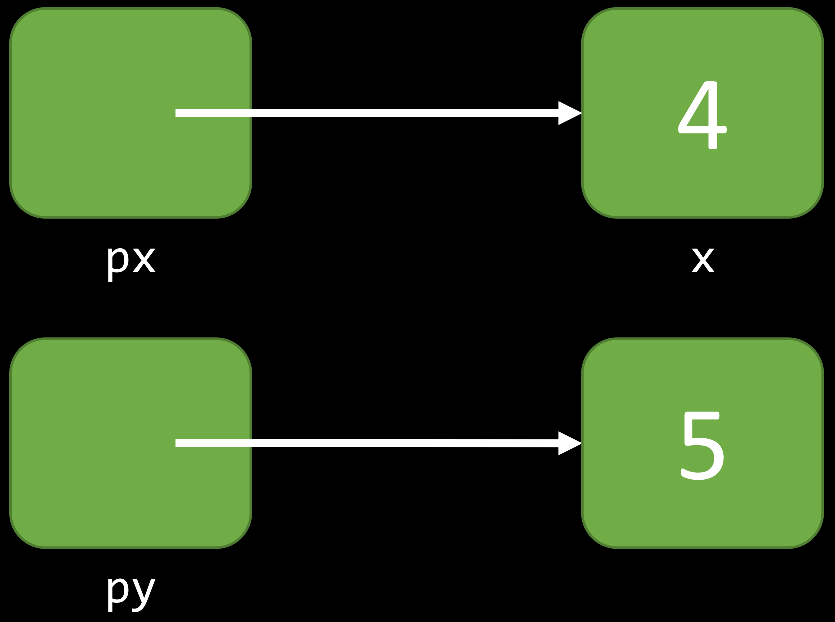
Week 3

```
int main(void)
{
    int x = 4;
    int *px = &x;
    int *py = malloc(sizeof(int));
    *py = 5;
}
```



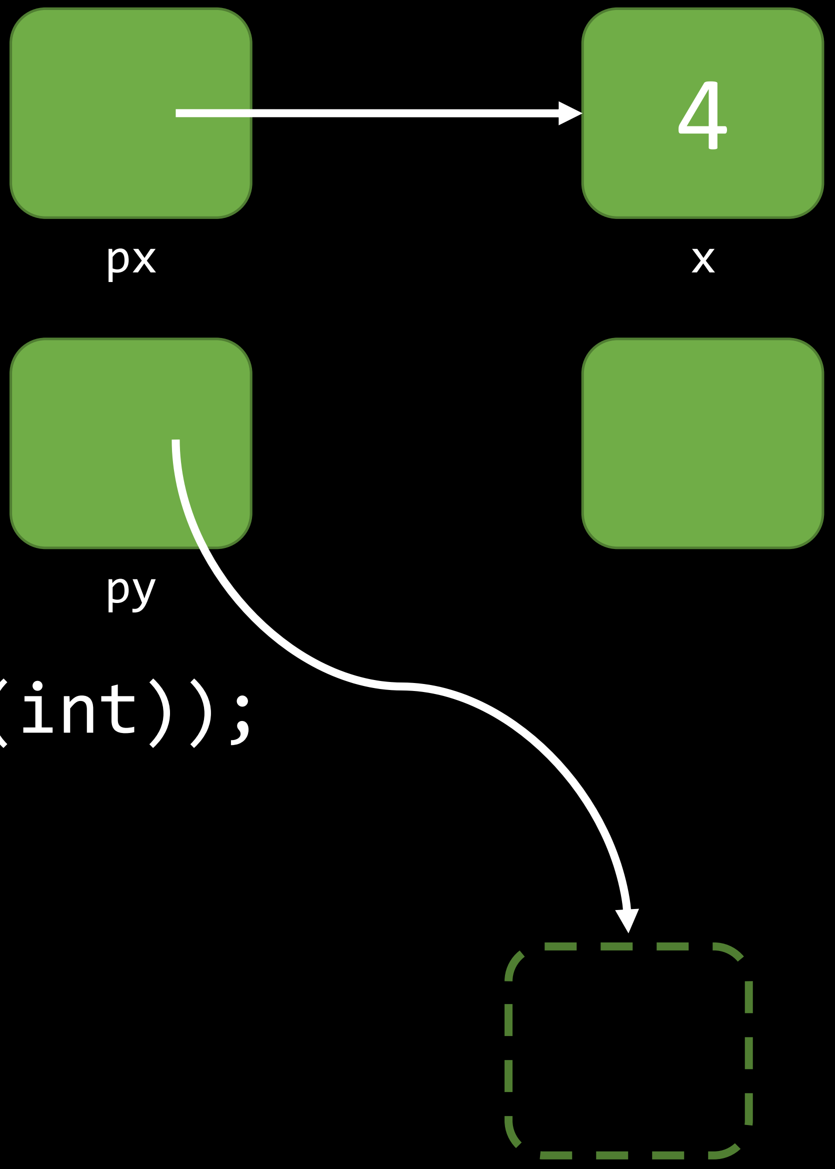
Week 3

```
int main(void)
{
    int x = 4;
    int *px = &x;
    int *py = malloc(sizeof(int));
    *py = 5;
}
```



Week 3

```
int main(void)
{
    int x = 4;
    int *px = &x;
    int *py = malloc(sizeof(int));
    py = 5;
}
```



Week 3

- Valgrind
  - Tool that we can use to spot memory leaks in our program.
  - Tells about any invalid thing we do with program’s memory.

Week 3

- Valgrind
  - Tool that we can use to spot memory leaks in our program.
  - Tells about any invalid thing we do with program's memory.
- Buffer Overflow
  - Integer overflow occurs when we try and store an integer larger than we are capable of storing.
  - Buffer overflow occurs when we try and store a string larger than we've set aside space for.
  - Can be used to malicious effect.

Week 3

- Memory
  - You can think of memory as a huge array of bytes.
  - Divided into two main segments, the *stack* and the *heap.*
    - Variables that you give a name to normally live on the stack.
    - Memory that you allocate dynamically lives on the heap.
  - These two segments are actually the same.
  - Possible segfault if they collide into each other.

Week 4

- File Operations
  - Special kind of structure used for abstracting a file on the file system.
  - `fopen` to obtain a file pointer (`FILE *`).
  - `fclose` when done working with it.
  - Reading from a file:
    - `fgetc`, `fgets`, `fread`, `fscanf`…
  - Writing to a file:
    - `fputc`, `fputs`, `fwrite`, `fprintf`…
  - Other operations:
    - `fseek`, `ftell`, `feof`, `ferror`…

Week 4

- Structures
  - C permits us to *encapsulate* data, by wrapping it up into a structure.
  - Group together related data into a single entity.
  - Dot operator to access a structure's members.
  - If we have pointers to structures, we use arrow (`->`) instead of dot, to dereference the pointer, then access the member.
  - `typedef` to give us cleaner type names.
  - Structures can be used, for instance, to organize image files.

Week 4

```
struct student
{
   char name[20];
   char house[20];
   int year;
   float gpa;
}
```

Week 4

```
struct student
{
   char name[20];
   char house[20];
   int year;
   float gpa;
}
```

```
struct student maria;

strcpy(maria.name, "Maria");
strcpy(maria.house, "Cabot");
maria.year = 2018;
maria.gpa = 5.00;
```

Week 4

- Linked Lists
  - Arrays suffer from a fixed-size limitation.
  - Lists grow and shrink with ease, but require dynamic memory.
  - Structure (node) with at least two members:
    - Data
    - A pointer to another structure in the same linked list (or to `NULL`).
  - Insertion and deletion can be constant time, O(1), operations.
  - Lookup/search is O(n), since we lose random access.
    - Start at the beginning of the chain, and work your way to the end.
    - Linear search.

Week 4

- Stacks
  - LIFO (last in, first out)
  - Linked list: You can only ever insert or delete from the head of the list.
  - Array: Keep track of most recently added element at all times.

Week 4

- Stacks
  - LIFO (last in, first out)
  - Linked list: You can only ever insert or delete from the head of the list.
  - Array: Keep track of most recently added element at all times.
- Queues
  - FIFO (first in, first out)
  - Linked list: You can only ever insert at the head of the list and delete from the tail of the list.
  - Array: Keep track of number of elements and “oldest” element at all times.

Week 4

- Trees
  - Node with normally at least three members:
    - Data
    - At least two pointers to other nodes lower in the tree (or to NULL)
  - Binary trees
  - Binary search trees
    - Lookup/search is **O(log n)** in a binary search tree.

Week 4

- Hash tables
  - Combination of a linked list and an array.
  - Use a hash function to get a value for your data.
  - Store in the linked list located at that index of the array.
  - Insertion can be constant time, O(1), operations.
  - Deletion and lookup are O(n).

Week 4

- Hash tables
  - Combination of a linked list and an array.
  - Use a hash function to get a value for your data.
  - Store in the linked list located at that index of the array.
  - Insertion can be constant time, O(1), operations.
  - Deletion and lookup are O(n).
- Tries
  - Special case of a tree.
  - Insertion and deletion can be constant time, O(1), operations.
  - Lookup/search is **O(1)** in a trie.

Week 5

- HTTP
  - Protocol for how clients should talk to servers, and vice versa.
  - A request includes (at minimum):
    - Request **method** (e.g. `GET`, `POST`).
    - Page (e.g. `/`).
    - HTTP Version (e.g. `HTTP/1.1`).
    - Website (e.g. `Host: www.facebook.com`).
  - Server responds back, with a status code (e.g. 200, 301, 404).

Week 5

- Status Codes

200	OK
301	Moved Permanently
302	Found
304	Not Modified
401	Unauthorized
403	Forbidden
404	Not Found
418	I’m a Teapot
500	Internal Server Error
503	Service Unavailable

Week 5

- IP Address
  - Number to identify addresses of devices on the Internet.
  - Formatted as #.#.#.#, where each # is in range 0 to 255.
  - DHCP (Dynamic Host Configuration Protocol) is used for computers to acquire an IP address.
  - Tools like `traceroute` let us inspect the path from client to server.

Week 5

- IP Address
  - Number to identify addresses of devices on the Internet.
  - Formatted as #.#.#.#, where each # is in range 0 to 255.
  - DHCP (Dynamic Host Configuration Protocol) is used for computers to acquire an IP address.
  - Tools like `traceroute` let us inspect the path from client to server.

- TCP
  - Transfer Control Protocol.
  - Port number corresponds to a service (e.g. 80, 443, 587).
  - Sends data in what are essentially numbered packets (1/4, 2/4…)

Week 5

- HTML
  - Hypertext Markup Language.
  - Describes the structure of a webpage, and contains the content for that page.
  - Nested start tags and closing tags (e.g. `<body>`, `</body>`) to delineate areas.

Week 5

- HTML
  - Hypertext Markup Language.
  - Describes the structure of a webpage, and contains the content for that page.
  - Nested start tags and closing tags (e.g. `<body>`, `</body>`) to delineate areas.

- CSS
  - Cascading Style Sheets.
  - Describe the aesthetics of web pages.
  - Selectors and attributes allow us to selectively modify only specific content on our page, rather than modify writ large.

Good luck!

Slides are available on the course website.