Comp 11 Lectures

Dr. Mike Shah

Tufts University

August 9, 2017

Please do not distribute or host these slides without prior permission.

Wrap Up

Comp 11 - Pre-Class warm up

Today's Agenda

- Pizza–come to class hungry!
- Return Finals
- Course wrap up
- Learn about C
- Learn about Python
- Share what you did for final project
- (Optional) Class Picture

Lecture

Hidden Figures

Figure 1: Katherine Johnson, Dorothy Vaughan, and Mary Jackson, were part of NASA's team of human "computers." They were a group of programmers and mathematicians who helped send John Glenn around the Earth. The 2016 Hollywood movie is excellent in my opinion.

Course Wrap up

Summary

- We learned modern C++
- Completed 6 homework assignments, 14 labs, and 15 in-class activities
- Got a preview of Data Structures (Comp 15), and Algorithms (Comp 160).
- You should be proud!
- I recommend continuing to learn a little bit of c++ each day–keep practicing!
  - Lynda, pluralsight, learncpp, cpluscplus, and youtube!

Programming in C

What is C?

- The C Programming language was created by Denis Ritchie.
- Popular *Programming in C* book is by Brian Kernighan and Ritchie, which is still a good read today! (Referred to as K and R book)
- C is essentially C++, without the object-oriented parts.
- It was one of the original systems programming languages.

Hello World from C

```
1  // The C library for input and output
2  #include <stdio.h>
3
4  int main(){
5
6    // Equivalent to std::cout statement
7    printf("Hello world!\n");
8
9    return 0;
10 }
```

Listing 1: Back to where we first started this time in C!

Compiling Hello World

- When we compile, we use only 'clang' without the ++.
- clang hello.c -o hello
- This means we are explicitly compiling C code only. No C++ features are available.

Similarities

- So it looks quite familiar to C++!
- What has changed from C++ are the libraries we call functions from.
- C++ was originally built on top of C.
- We can use in our C++ code, any of the C libraries (remember to compile with clang++).

Memory is King

- In C, we are often working very closely with memory.
- We use **malloc** instead of **new** - to allocate memory
- We use **free** instead of **delete** - to reclaim memory.
- Mixing the two will cause compiler problems.

Malloc and Free Example

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(){
5      int size = 100;
6      int* myIntArray = (int*)malloc(100*sizeof(int));
7
8      for(int i =0; i < size; ++i){
9          myIntArray[i] = i;
10         printf("%d ", myIntArray[i]);
11     }
12
13     free(myIntArray);
14
15     return 0;
16 }
```

Listing 2: Equivalent to new and delete in C++ but they should not be mixed

Memory functions

- C offers some powerful functions for quickly operating on memory.
- memset
- memcpy

memset example

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(){
5
6      int n = 100;
7      int myArray[n];
8      // Set each byte at a time to 0
9      memset(myArray,0, n*sizeof(myArray[0]));
10
11     for(int i =0; i < n; ++i){
12         printf("%d ", myArray[i]);
13     }
14
15     return 0;
16 }
```

Listing 3: Set one byte at a time to 0

Why use C over C++

- Generally a smaller footprint (executable sizes are smaller)
- This is helpful if we are programming embedded devices (think watches, a refrigerator, or a microwave, or items where we have limited storage).
- C has a more 1 to 1 ratio with assembly language, where you have even more control of how machine uses your code.

Here is what assembly looks like by the way

```
; example from http://asm.sourceforge.net/intro/hello.html
section     .text
global      _start          ;must be declared for linker (ld)

_start:                     ;tell linker entry point
    mov     edx,len         ;message length
    mov     ecx,msg         ;message to write
    mov     ebx,1           ;file descriptor (stdout)
    mov     eax,4           ;system call number (sys_write)
    int     0x80            ;call kernel
    mov     eax,1           ;system call number (sys_exit)
    int     0x80            ;call kernel
section     .data
msg     db  'Hello, world!',0xa  ;our dear string
len     equ $ - msg                ;length of our dear string
```

Listing 4: Assembly actually maps almost 1 to 1 with machine code

Python

What is Python?

- Python is a high-level programming language created by Guido van Rossum.
- It is general purpose, in that any sort of program can be made.
- It is more a scripting language, that is, programs do not compile to executables.
- There is Python 2.7.X and Python 3 most actively used. (Similar to how we have C++ and C++11/14/17)
- Today we will be discussing **Python 3**, which is slightly different than 2.7, but will long term likely be the more dominant language.

Python - Follow Along

https://repl.it/languages/python

Python - Hello World

```
1 print "Hello World\n"
```

Listing 5: Hello world in python

- Wait-should we have more source code here, this slide looks empty
- Nope!
- Some trade offs on the next slides.

Python - Guessing Game in a Slide!

```
1  # Pound is the comment symbol
2  import random # include a library
3  secret = randint(1, 10) # Generate a random number
4  guess = -1
5
6  # Create our while loop
7  while guess != secret;
8      guess = raw_input()
9
10     if guess < secret:
11       print "guess higher"
12     else:
13       print "guess lower"
14
15 print "You got it!
```

Listing 6: Guessing Game

A few observations

- No curly braces {}'s
- Instead, Python uses whitespace to create code blocks terminated with a colon to determine scope
- Also, less parentheses ()'s or semi-colons
- The language is very clean!

Python - The List

```
1  myList = []
2
3  myList.append(1)
4  myList.append(2)
5  myList.append(3)
6
7  print myList
8
9  myList.pop()
10 myList.pop()
11
12 print myList
```

Listing 7: The fundamental data structure in Python

Python - List Comprehension

```
1 numbers = [1,2,3,4,5]
2
3 # 3 lines of code to make a new list
4 squaresList = []
5 for n in numbers:
6   squaresList.append(n*n)
7
8 # 1 line of code to make a new list
9 squaresListComprehension = [n*n for n in numbers]
```

Listing 8: Generate a list

Python - Plots

```
1  import matplotlib.pyplot as plt
2  import math
3
4  fig, ax = plt.subplots()
5
6  logGrowth = []
7  linearGrowth = []
8  quadraticGrowth = []
9  for i in range(1,1000):
10   linearGrowth.append(i)
11   logGrowth.append(math.log(i,2))
12   quadraticGrowth.append(i*i);
13
14 ax.plot(logGrowth)
15 ax.plot(linearGrowth)
16 ax.plot(quadraticGrowth)
17
18 plt.show()
```

Listing 9: Generate a list

Python - Functions

```
1 # Define a function (return type inferred at run-time)
2 def square(x):
3   return x*x
4
5 # Create a list of 10 numbers
6 numbers = range(0,10)
7
8 # Use a list comprehension
9 squares = [square(n) for n in numbers]
10
11 # Print out the result
12 print squares
```

Listing 10: Create a simple function

Python trade offs part 1

- Python is extremely productive, in that there exist many libraries(i.e. package) to do the heavy lifting for us.
- The language was designed 20+ years after C, so there was time to innovate. C did its job for providing a higher level language in a resource constrained environment.
- There is no need to manage memory in Python (It is a memory managed language)
- There is no need to even declare the data type! (It is dynamically type, as opposed to statically typed–the interpreter figures out what you mean)
- Python also has a nice REPL (Read-evaluate-print-loop) that lets you rapidly prototype code and ideas.
- You can do Object-Oriented program in Python just like in C++!

Python trade offs part 2

- Typically, a Python program will run more slowly because it is interpreted.
- This means there is some overhead of the Python Interpreter that reads each line of python code you wrote over and over.
- There is no translation to 1's and 0's that the computer can quickly munch on.

In-Class Activity

- Quick 1-2 sentence pitch of your final project
  - State your name (so your classmates remember you!)
  - 1-2 sentence project pitch.

The End! (Of Comp 11)

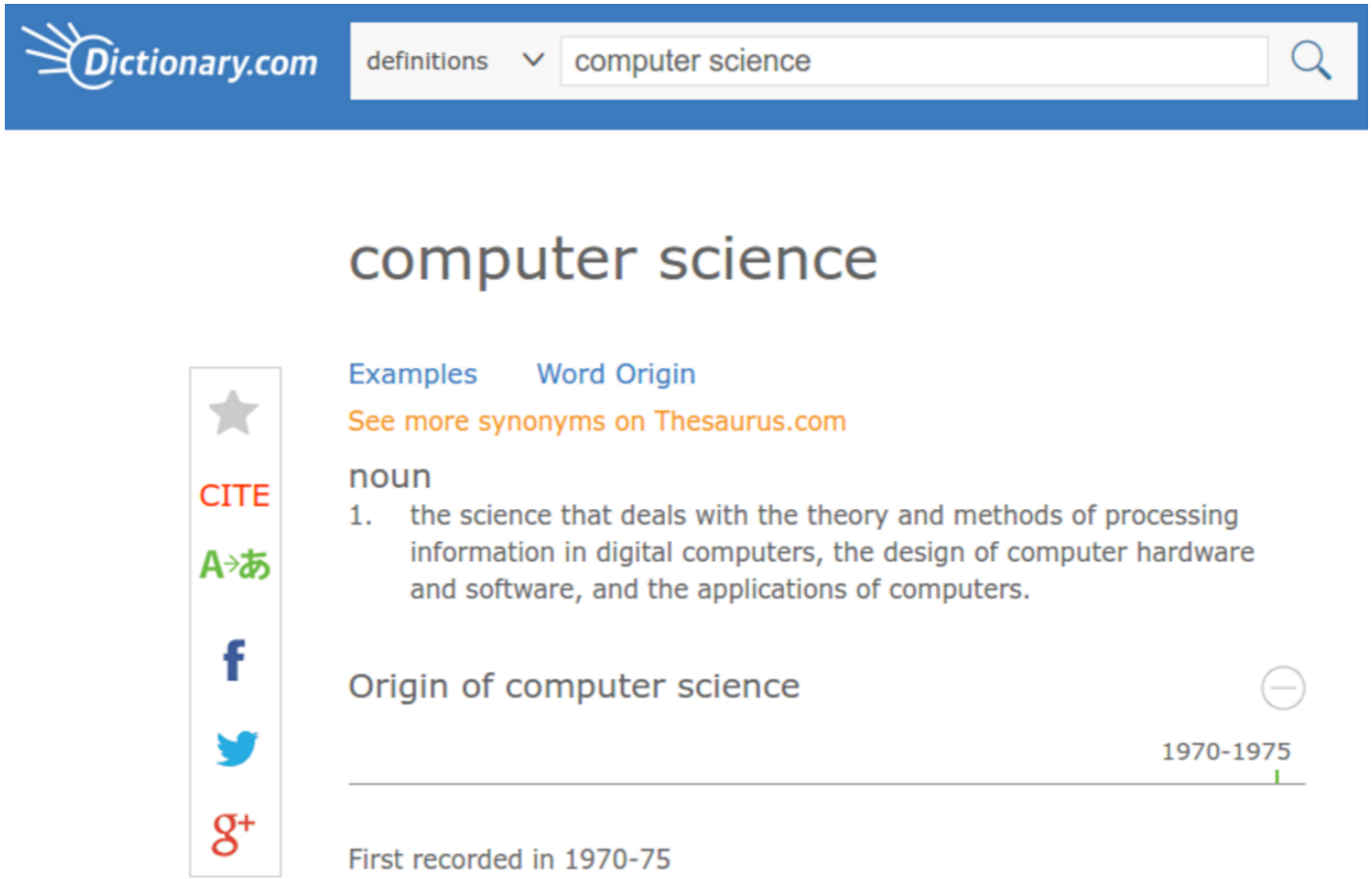

Figure 2: We made it!

Glossary