Intro, UNIX, Bash, C

CS 5006, 5007: C, Algorithms and Systems

Adrienne Slaughter, Joe Buck

Northeastern University

January 9, 2019

1. Intro to CS 5007

2. Course Overview

3. Intro to Architecture

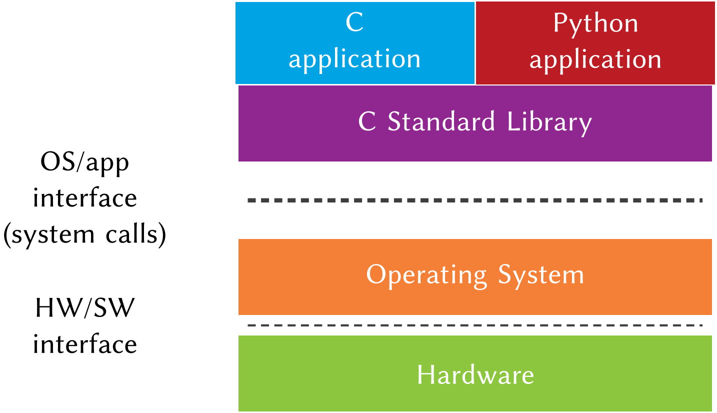
4. C

5. Input/Output (IO) in C
   - Command line IO
     - Data Hierarchy

Section 1

Intro to CS 5007

The Big Picture: What is a System?



Agenda

- Course Overview/Structure
- Introduction to C programming
- Set up programming environment: VirtualBox

CS 5006, CS 5007: Algorithms and Systems

Lecture time: Tuesdays from 9:00am — 12:00pm in 225 Terry, Room 306

- Instructors:
    - Adrienne Slaughter (a.slaughter@northeastern.edu)
        - TBD
        - By appointment
    - Joe Buck (j.buck@northeastern.edu)
        - TBD

CS 5006: Algorithms

TAs:

- Bicheng Xu
- Chenxi Liu
- Jackie Tseng
- Chi Moua
- Wes Florence
- TBD

Course material:	Course website. Algorithms Unlocked (Cormen) Systems: A Programmer's Perspective (O'Halloran and Bryant)
Course discussion board:	Piazza
Course assignment submission:	CCIS GitHub
Assignment grades:	NEU Blackboard

What is CS 5006 and 5007, Spring 2019?

C and Systems, with algorithms

- Intended for students in the ALIGN MS in CS program

Course Goals

1. Familiarity with computer systems

What is CS 5006 and 5007, Spring 2019?

C and Systems, with algorithms

- Intended for students in the ALIGN MS in CS program

Course Goals

1. Familiarity with computer systems
2. Proficiency with C programming

What is CS 5006 and 5007, Spring 2019?

C and Systems, with algorithms

- Intended for students in the ALIGN MS in CS program

Course Goals

1. Familiarity with computer systems
2. Proficiency with C programming
3. Practical skills with *nix systems

What is CS 5006 and 5007, Spring 2019?

C and Systems, with algorithms

- Intended for students in the ALIGN MS in CS program

Course Goals

1. Familiarity with computer systems
2. Proficiency with C programming
3. Practical skills with *nix systems
4. Apply algorithmic analysis to implementation

CS 5006/5007 Spring 2019: Course Outcomes

At the end of this course, you should be able to:

- Navigate, edit text files, compile and run programs on a command line
- Describe the architecture of a computer
- Write, debug and test C programs
- Describe how multiple threads, processes, and synchronization works.
- Describe client-server networking

(Expected) Course Progression

- Week 1: Getting Started with C programming.

(Expected) Course Progression

- Week 1: Getting Started with C programming.
- Week 2: Arrays, memory. Searching and sorting.

(Expected) Course Progression

- Week 1: Getting Started with C programming.
- Week 2: Arrays, memory. Searching and sorting.
- Week 3: Structs, pointers.

(Expected) Course Progression

- Week 1: Getting Started with C programming.
- Week 2: Arrays, memory. Searching and sorting.
- Week 3: Structs, pointers.
- Week 4: Data Structures.

(Expected) Course Progression

- Week 1: Getting Started with C programming.
- Week 2: Arrays, memory. Searching and sorting.
- Week 3: Structs, pointers.
- Week 4: Data Structures.
- Week 5: Trees and Graphs.

(Expected) Course Progression

- Week 1: Getting Started with C programming.
- Week 2: Arrays, memory. Searching and sorting.
- Week 3: Structs, pointers.
- Week 4: Data Structures.
- Week 5: Trees and Graphs.
- Week 6: More Trees and Graphs.

(Expected) Course Progression

- Week 1: Getting Started with C programming.
- Week 2: Arrays, memory. Searching and sorting.
- Week 3: Structs, pointers.
- Week 4: Data Structures.
- Week 5: Trees and Graphs.
- Week 6: More Trees and Graphs.
- Week 7: C libraries and advanced data structures.

(Expected) Course Progression

- Week 1: Getting Started with C programming.
- Week 2: Arrays, memory. Searching and sorting.
- Week 3: Structs, pointers.
- Week 4: Data Structures.
- Week 5: Trees and Graphs.
- Week 6: More Trees and Graphs.
- Week 7: C libraries and advanced data structures.
- Week 8: Midterm/Final

(Expected) Course Progression

- Week 9: Transition to systems: Bash programming, file systems.

(Expected) Course Progression

- Week 9: Transition to systems: Bash programming, file systems.
- Week 10: Memory hierarchy, Computer organization

(Expected) Course Progression

- Week 9: Transition to systems: Bash programming, file systems.
- Week 10: Memory hierarchy, Computer organization
- Week 11: Multi-threading.

(Expected) Course Progression

- Week 9: Transition to systems: Bash programming, file systems.
- Week 10: Memory hierarchy, Computer organization
- Week 11: Multi-threading.
- Week 12: Synchronization and deadlocks.

(Expected) Course Progression

- Week 9: Transition to systems: Bash programming, file systems.
- Week 10: Memory hierarchy, Computer organization
- Week 11: Multi-threading.
- Week 12: Synchronization and deadlocks.
- Week 13: Networking.

(Expected) Course Progression

- Week 9: Transition to systems: Bash programming, file systems.
- Week 10: Memory hierarchy, Computer organization
- Week 11: Multi-threading.
- Week 12: Synchronization and deadlocks.
- Week 13: Networking.
- Week 14: Advanced topics in Systems.

(Expected) Course Progression

- Week 9: Transition to systems: Bash programming, file systems.
- Week 10: Memory hierarchy, Computer organization
- Week 11: Multi-threading.
- Week 12: Synchronization and deadlocks.
- Week 13: Networking.
- Week 14: Advanced topics in Systems.
- Week 15: Final Project due

Overview of Assignments

- Week 1: Getting Started with C programming.

Overview of Assignments

- Week 1: Getting Started with C programming.
- Week 2: Iterating through arrays, searching and sorting.

Overview of Assignments

- Week 1: Getting Started with C programming.
- Week 2: Iterating through arrays, searching and sorting.
- Week 3: Practicing with structs and pointers.

Overview of Assignments

- Week 1: Getting Started with C programming.
- Week 2: Iterating through arrays, searching and sorting.
- Week 3: Practicing with structs and pointers.
- Week 4: Building basic data structures in C.

Overview of Assignments

- Week 1: Getting Started with C programming.
- Week 2: Iterating through arrays, searching and sorting.
- Week 3: Practicing with structs and pointers.
- Week 4: Building basic data structures in C.
- Week 5: Building and working with graphs and trees. Possibly Huffman encoding.

Overview of Assignments

- Week 1: Getting Started with C programming.
- Week 2: Iterating through arrays, searching and sorting.
- Week 3: Practicing with structs and pointers.
- Week 4: Building basic data structures in C.
- Week 5: Building and working with graphs and trees. Possibly Huffman encoding.
- Week 6: Continuation of Huffman.

Overview of Assignments

- Week 1: Getting Started with C programming.
- Week 2: Iterating through arrays, searching and sorting.
- Week 3: Practicing with structs and pointers.
- Week 4: Building basic data structures in C.
- Week 5: Building and working with graphs and trees. Possibly Huffman encoding.
- Week 6: Continuation of Huffman.
- Week 7: Advanced data structures (library), testing.

Overview of Assignments

- Week 1: Getting Started with C programming.
- Week 2: Iterating through arrays, searching and sorting.
- Week 3: Practicing with structs and pointers.
- Week 4: Building basic data structures in C.
- Week 5: Building and working with graphs and trees. Possibly Huffman encoding.
- Week 6: Continuation of Huffman.
- Week 7: Advanced data structures (library), testing.
- Week 8: Midterm/Final

Overview of Assignments

- Week 9: Advanced data structures (library), testing (cont)

Overview of Assignments

- Week 9: Advanced data structures (library), testing (cont)
- Week 10: File system crawler/indexer.

Overview of Assignments

- Week 9: Advanced data structures (library), testing (cont)
- Week 10: File system crawler/indexer.
- Week 11: Multithreaded crawler.

Overview of Assignments

- Week 9: Advanced data structures (library), testing (cont)
- Week 10: File system crawler/indexer.
- Week 11: Multithreaded crawler.
- Week 12: Build query processor.

Overview of Assignments

- Week 9: Advanced data structures (library), testing (cont)
- Week 10: File system crawler/indexer.
- Week 11: Multithreaded crawler.
- Week 12: Build query processor.
- Week 13: Final Project: Build webserver.

Overview of Assignments

- Week 9: Advanced data structures (library), testing (cont)
- Week 10: File system crawler/indexer.
- Week 11: Multithreaded crawler.
- Week 12: Build query processor.
- Week 13: Final Project: Build webserver.
- Week 14:

Overview of Assignments

- Week 9: Advanced data structures (library), testing (cont)
- Week 10: File system crawler/indexer.
- Week 11: Multithreaded crawler.
- Week 12: Build query processor.
- Week 13: Final Project: Build webserver.
- Week 14:
- Week 15: Final Project due

Course Logistic

Course will be graded based upon:

- Homework assignments: 60%
  - Exercises
  - Problems
  - Reflection
- Midterm: 15%
- Final Project: 15%

Late Turn in Policy

- All assignments are due by midnight on the assigned date

Late Turn in Policy

- All assignments are due by midnight on the assigned date
- Late assignments get a 0

Late Turn in Policy

- All assignments are due by midnight on the assigned date
- Late assignments get a 0
- If you have a meaningful reason for delay (e.g., illness)— come and talk to me

Late Turn in Policy

- All assignments are due by midnight on the assigned date
- Late assignments get a 0
- If you have a meaningful reason for delay (e.g., illness)— come and talk to me
- If you have a request for an extension for some other reasonable reason, you must talk to me **in advance**.

Collaboration and Academic Integrity

- You can talk to others about the ideas, but all write-ups and answers must be your own.
- If in doubt, cite.
  - Make a note of who you talked to or a website you looked at.

Course Logistics: Exam

- Closed book

Course Logistics: Exam

- Closed book
- Covers the entire course so far (all lectures)

Course Logistics: Exam

- Closed book
- Covers the entire course so far (all lectures)
- 1 page of notes? Maybe. (Probably)

Course Materials

Website

- https://course.ccs.neu.edu/cs5007sp19-seattle

Resources:

- Algorithms Unlocked (Cormen)
- Computer Systems: A Programmer's Perspective, 3rd Edition, Bryant and O'Halloran
- Will be posted on https://course.ccs.neu.edu/cs5007sp19-seattle/resources.html as the semester progresses
- Cormen, Leiserson and Rivest is a classic algorithm text
- The Algorithm Design Manual (Skiena) is also great

Tips for Success

- Read the assigned material
- Attempt to solve additional problems
- Attend lectures
- Talk to the course staff
- Keep up
- Talk to each other

Questions

Questions?

Code for the next examples

```c
#include<stdio.h>

int main()
{
  printf("Hello world\n");
  return 0;
}

```

Listing 1: “hello.c”

Compiling and running

```
[ahslaughter@adriennes-mbp:~]\$ gcc hello.c
[ahslaughter@adriennes-mbp:~]\$ ./a.out

```

Listing 2: To compile and run

```
[ahslaughter@adriennes-mbp:~]\$ gcc hello.c -o hello
[ahslaughter@adriennes-mbp:~]\$ ./hello

```

Listing 3: To compile and run with named outfile

Makefile

```
all: hello

hello: hello.c
  gcc —o hello hello.c

run: hello
  ./hello

clean:
  rm *.o  hello *.a
```

We have 4 ***targets*** listed: all, hello, run, and clean.

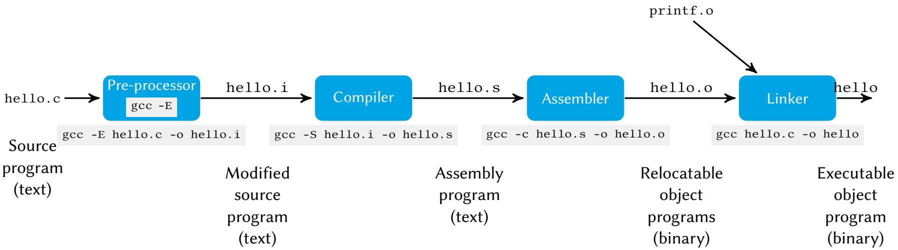
Compiling and running

```
[ahslaughter@adriennes-mbp:~]\$ make run

```

Listing 4: Compiling and running with Make

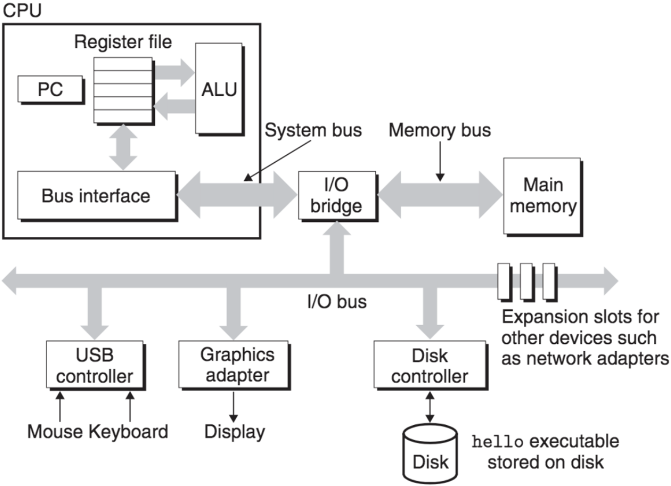
Because we've set up the targets in the Makefile, running `make run` ensures that everything is compiled if it needs to be (but not if it doesn't!), and then runs the program.



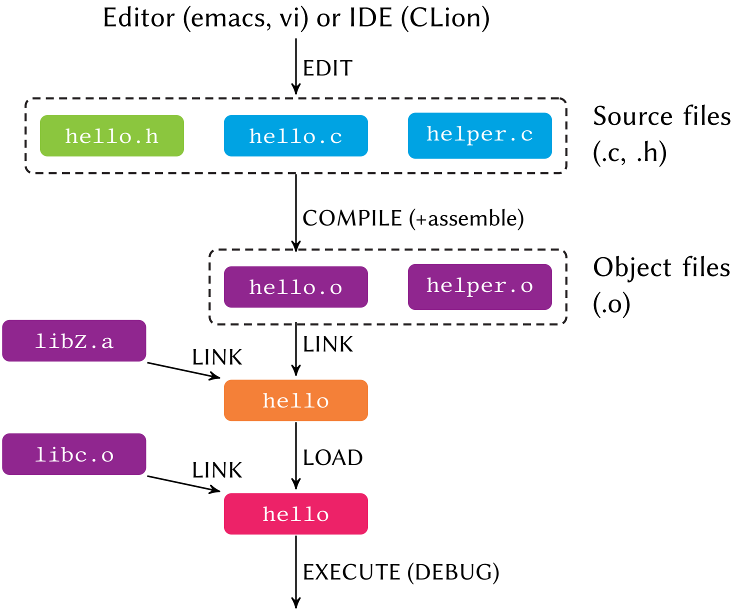
```
gcc -E hello.c -o hello.i
gcc -S hello.i -o hello.s
gcc -c hello.s -o hello.o
gcc hello.c -o hello
```

Computer Organization

What happens when we run our program?



C programs with multiple files


Editor (emacs, vi) or IDE (CLion)
EDIT
hello.h
hello.c
helper.c
Source files
(.c, .h)
COMPILE (+assemble)
hello.o
helper.o
Object files
(.o)
libZ.a
LINK
LINK
hello
libc.o
LINK
LOAD
hello
EXECUTE (DEBUG)

Section 4

C

C Refresher

- Header files: `*.h`
  - Holds your function prototypes
- C files: `*.c`
  - Holds your C code
- make file: `makefile`
  - Sets up your build
  - `make <target>`
  - Determines if relevant files have changed or not, and rebuilds accordingly

Make

```
all: quiz4

quiz4: quiz4.h quiz4.c quiz4_test.c
  gcc quiz4.c quiz4_test.c -o quiz4

.PHONY: clean
clean:
  rm -f quiz4

```

Listing 5: Sample `makefile`

Section 5

Input/Output (IO) in C

Subsection 1

Command line IO

Streams

Streams

- All input and output uses ***streams***

Streams

- All input and output uses ***streams***
- A stream is a sequence of characters organized into a line

Streams

- All input and output uses ***streams***
- A stream is a sequence of characters organized into a line
- The line ends with a newline character ('`\0`')

Streams

- All input and output uses ***streams***
- A stream is a sequence of characters organized into a line
- The line ends with a newline character ('`\0`')
- Three streams are connected to a program automatically when it runs:
  - ***standard input*** by default is connected to the keyboard
  - ***standard output*** by default is connected to the screen (terminal)
  - ***standard error*** by default is connected to the screen. (Error messages are output the error stream)

Streams

- All input and output uses ***streams***
- A stream is a sequence of characters organized into a line
- The line ends with a newline character ('`\0`')
- Three streams are connected to a program automatically when it runs:
  - ***standard input*** by default is connected to the keyboard
  - ***standard output*** by default is connected to the screen (terminal)
  - ***standard error*** by default is connected to the screen. (Error messages are output the error stream)
- Opening a file returns a pointer to a FILE, which includes a ***file descriptor***, which is an index into the OS array ***open file table***.

Standard Input and Output

- `getchar`
- `putchar`
- `gets`
- `puts`
- `printf`
- `scanf`

Reading/Writing to Command Line

Function Prototype	Function Description

Reading/Writing to Command Line

Function Prototype	Function Description
`int getchar(void)`	Input the next character from the standard input and return it as an integer.

Reading/Writing to Command Line

Function Prototype	Function Description
`int getchar(void)`	Input the next character from the standard input and return it as an integer.
`int putchar(int c)`	Print the character stored in c.

Reading/Writing to Command Line

Function Prototype	Function Description
`int getchar(void)`	Input the next character from the standard input and return it as an integer.
`int putchar(int c)`	Print the character stored in c.
`int puts(const char *s)`	Print the string s followed by a newline

Reading/Writing to Command Line

Function Prototype	Function Description
`int getchar(void)`	Input the next character from the standard input and return it as an integer.
`int putchar(int c)`	Print the character stored in c.
`int puts(const char *s)`	Print the string s followed by a newline
`void printf(char *format, …)`	Print the params formatted per the format.

Reading/Writing to Command Line

Function Prototype	Function Description
`int getchar(void)`	Input the next character from the standard input and return it as an integer.
`int putchar(int c)`	Print the character stored in c.
`int puts(const char *s)`	Print the string s followed by a newline
`void printf(char *format, …)`	Print the params formatted per the format.
`void scanf(char *format, …)`	Read input into the given variables

Example: `getchar` and `puts`

```c
#include<stdio.h>

int main(){
  char c, sentence[80];
  int i=0;

  puts("Enter a line of text: ");
  while ((c = getchar()) != '\n'){
    sentence[i++] = c;
  }

  sentence[i] = '\0';
  puts("\nThe line entered was: ");
  puts(sentence);
  return 0;
}
```

Listing 6: `puts` and `getchar`

Example: `scanf` and `putchar`

```c
#include<stdio.h>

void reverse(char *);

int main(){
  char sentence[80];

  printf("Enter a line of text: \n");
  scanf("%s", sentence);

  printf("\nThe line printed backwards is: \n");
  reverse(sentence);
  printf("\n");
  return 0;
}

void reverse(char *s){
  if (s[0] == '\0'){
    return;
  }
  else{
    reverse(&s[1]);
    putchar(s[0]);
```

Are we missing something?

We had `printf` & `scanf`.

Are we missing something?

We had `printf` & `scanf`.

We had `getchar` & `putchar`.

Are we missing something?

We had `printf` & `scanf`.

We had `getchar` & `putchar`.

But I only showed you `puts`, no `gets`.

Why??

Are we missing something?

We had `printf` & `scanf`.

We had `getchar` & `putchar`.

But I only showed you `puts`, no `gets`.

Why??

There is a `gets`:

`char *gets(char *s)`: Input characters from the standard input into the array s until a newline or end-of-file character is encountered. A terminating NULL is appended to the array.

Are we missing something?

We had `printf` & `scanf`.

We had `getchar` & `putchar`.

But I only showed you `puts`, no `gets`.

Why??

There is a `gets`:

`char *gets(char *s)`: Input characters from the standard input into the array s until a newline or end-of-file character is encountered. A terminating NULL is appended to the array.

- We don't know how big the input is, and it can overflow the buffer.

C Command line I/O Summary

- Can get and put chars with `getchar` and `putchar`
- Can print and scan formatted strings with `printf` and `scanf`
- Can print strings with `puts`
- Can, but shouldn't, get strings with `gets`

Miscellaneous

- `fgetc(stdin)` is equivalent to `getchar()`.
- `fputc('a', stdout)` is equivalent to `putchar('a')`.

Data Hierarchy

A file

Sally	Purple		
Tom	Orange		
Joe	Green		
Callie	Yellow		

Data Hierarchy

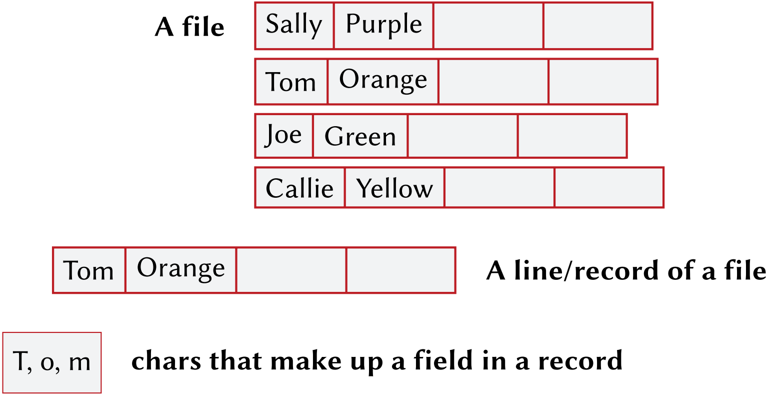
A file

Sally	Purple		
Tom	Orange		
Joe	Green		
Callie	Yellow		

Tom	Orange		

A line/record of a file

Data Hierarchy



A file

Sally	Purple		
Tom	Orange		
Joe	Green		
Callie	Yellow		

Tom	Orange		

A line/record of a file

T, o, m

chars that make up a field in a record

Data Hierarchy

A file

Sally	Purple		
Tom	Orange		
Joe	Green		
Callie	Yellow		

Tom	Orange		

A line/record of a file

T, o, m

chars that make up a field in a record

01010100

a byte that represents a char in a field

Data Hierarchy

A file

Sally	Purple		
Tom	Orange		
Joe	Green		
Callie	Yellow		

Tom	Orange		

A line/record of a file

T, o, m **chars that make up a field in a record**

01010100 **a byte that represents a char in a field**

0 **a bit in the byte**

1. Intro to CS 5007

2. Course Overview

3. Intro to Architecture

4. C

5. Input/Output (IO) in C
   - Command line IO
     - Data Hierarchy