The Edit–Compile–Run Cycle

CS 211

Road map

The Edit–Compile–Run Cycle

Compilation

Road map

Compilation

The Unix shell

The Edit–Compile–Run Cycle

Road map

Compilation

The Unix shell

Building with Make

The Edit–Compile–Run Cycle

Road map

Compilation

The Unix shell

Building with Make

Getting & building starter code

The Edit–Compile–Run Cycle

Road map

Compilation

The Unix shell

Building with Make

Getting & building starter code

Appendix: Numeral systems

The Edit–Compile–Run Cycle

Road map

Compilation

The Unix shell

Building with Make

Getting & building starter code

Appendix: Numeral systems

Appendix: More Make

The Edit–Compile–Run Cycle

Up next

The Edit–Compile–Run Cycle

Compilation

The Unix shell

Building with Make

Getting & building starter code

Appendix: Numeral systems

Appendix: More Make

So you've written a C program:

```c
#include <stdio.h>

int main(void)
{
    printf("Hello, CS 211!\n");
}
```

What now?

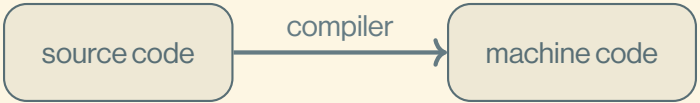
Compilation

We need to translate our program from

- source code (human readable, *e.g.,* C or Swift)

to

- machine code (machine executable, *e.g.,* x86-64 or ARM).



What does machine code look like? (1/3)

85	72	137	229	72	131	236	16
72	141	61	55	0	0	0	176
0	232	14	0	0	0	49	201
137	69	252	137	200	72	131	196
16	93	195					

(Each byte value ranges from 0 to 255.)

What does machine code look like? (2/3)

55	48	89	E5	48	83	EC	10
48	8D	3D	37	00	00	00	B0
00	E8	0E	00	00	00	31	C9
89	45	FC	89	C8	48	83	C4
10	5D	C3					

(Each byte value ranges from 0x00 to 0xFF.)

What does machine code look like? (2/3)

55	48	89	E5	48	83	EC	10
48	8D	3D	37	00	00	00	B0
00	E8	0E	00	00	00	31	C9
89	45	FC	89	C8	48	83	C4
10	5D	C3					

(Each byte value ranges from 0x00 to 0xFF.)

(These numbers are written in base 16, a/k/a hexadecimal, which uses letters A–F as digits with values 10–15.)

What does machine code look like? (3/3)

```
55                      pushq %rbp
48 89 e5                movq  %rsp,     %rbp
48 83 ec 10             subq  $16,      %rsp
48 8d 3d 37 00 00 00    leaq  55(%rip), %rdi
b0 00                   movb  $0,       %al
e8 0e 00 00 00          callq 14
31 c9                   xorl  %ecx,     %ecx
89 45 fc                movl  %eax,     -4(%rbp)
89 c8                   movl  %ecx,     %eax
48 83 c4 10             addq  $16,      %rsp
5d                      popq  %rbp
c3                      retq
```

(Machine code printed as assembly language mnemonics.)

Up next

Compilation

The Unix shell

Building with Make

Getting & building starter code

Appendix: Numeral systems

Appendix: More Make

Using Unix

For the first few weeks of class, we are going to develop and test our programs under Unix.

Using Unix

For the first few weeks of class, we are going to develop and test our programs under Unix.

Unix A style of multi-user operating system with half a century of development. (Modern variants include Linux and macOS.)

Using Unix

For the first few weeks of class, we are going to develop and test our programs under Unix.

Unix A style of multi-user operating system with half a century of development. (Modern variants include Linux and macOS.)

shell The main program for controlling a Unix computer, using text commands.

Using Unix

For the first few weeks of class, we are going to develop and test our programs under Unix.

Unix	A style of multi-user operating system with half a century of development. (Modern variants include Linux and macOS.)
shell	The main program for controlling a Unix computer, using text commands.
terminal	A program (or historically, device) for displaying text-based interactions with a Unix computer, often remote.

Advantages of the Unix shell (1/2)

Compared to point-and-click, you can say more with less:

```
$ mkdir backup
$ cp *.docx backup
```

Advantages of the Unix shell (1/2)

Compared to point-and-click, you can say more with less:

```
$ mkdir backup
$ cp *.docx backup

$ mkdir thumbs
$ for i in *.png
>   convert -geometry 128x128 "$i" "thumbs/$i"
> end
```

Advantages of the Unix shell (2/2)

You can automate repeated tasks by putting common sequences of commands in *shell scripts*:

```
#!/bin/sh

for dir in "$@"; do
  (
    cd "$dir"
    mkdir -p thumbs
    for file in *.png; do
      convert -geometry 128x128 \
          "$file" "thumbs/$file"
    done
  )
done
```

Compilation in the Unix shell

$

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
%
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
%
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
%
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
% emacs hello.c         # edit (new) file
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
% emacs hello.c         # edit (new) file
%
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
% emacs hello.c         # edit (new) file
% ls                    # list directory contents
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
% emacs hello.c         # edit (new) file
% ls                    # list directory contents
hello.c
%
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
% emacs hello.c         # edit (new) file
% ls                    # list directory contents
hello.c
% cc hello.c            # compile C program
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
% emacs hello.c         # edit (new) file
% ls                    # list directory contents
hello.c
% cc hello.c            # compile C program
%
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
% emacs hello.c         # edit (new) file
% ls                    # list directory contents
hello.c
% cc hello.c            # compile C program
% ls                    # list directory contents
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
% emacs hello.c         # edit (new) file
% ls                    # list directory contents
hello.c
% cc hello.c            # compile C program
% ls                    # list directory contents
a.out  hello.c
%
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
% emacs hello.c         # edit (new) file
% ls                    # list directory contents
hello.c
% cc hello.c            # compile C program
% ls                    # list directory contents
a.out  hello.c
% ./a.out               # run compiled program
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Compilation in the Unix shell

```
$ fish                  # switch to fish shell
% mkdir cs211           # make directory
% cd cs211              # change directory
% emacs hello.c         # edit (new) file
% ls                    # list directory contents
hello.c
% cc hello.c            # compile C program
% ls                    # list directory contents
a.out  hello.c
% ./a.out               # run compiled program
Hello, CS 211!
%
```

(In this week's lab exercises you'll do the setup necessary to get access to `fish` and other stuff you need.)

Up next

Compilation

The Unix shell

Building with Make

Getting & building starter code

Appendix: Numeral systems

Appendix: More Make

Build management

As programs get larger, builds get more complicated:

- More files to compile, in complex combinations
- Want to just recompile the changed files
- Different compilers/machines want different options and work differently

Build management

As programs get larger, builds get more complicated:

- More files to compile, in complex combinations
- Want to just recompile the changed files
- Different compilers/machines want different options and work differently

We'll use a software building system called Make to automate builds for us

The `Makefile`

Make is configured using a file called `Makefile`, which is a set of rules that say what you can build, what it's built from, and how.

The `Makefile`

Make is configured using a file called `Makefile`, which is a set of rules that say what you can build, what it's built from, and how.

The simplest possible `Makefile`:

```
hello: hello.c
	cc -o hello hello.c
```

The `Makefile`

Make is configured using a file called `Makefile`, which is a set of rules that say what you can build, what it's built from, and how.

The simplest possible `Makefile`:

```
hello: hello.c
	cc -o hello hello.c
```

(Meaning: To build `hello` from `hello.c`, run the command `cc -o hello hello.c`.)

Running a Make recipe

%

Running a Make recipe

```
% make hello
```

Running a Make recipe

```
% make hello
cc -o hello hello.c  # Make prints the commands
%
```

Running a Make recipe

```
% make hello
cc -o hello hello.c  # Make prints the commands
% make hello
```

Running a Make recipe

```
% make hello
cc -o hello hello.c  # Make prints the commands
% make hello         # and avoids unnecessary work
make: `hello' is up to date.
%
```

Running a Make recipe

```
% make hello
cc -o hello hello.c  # Make prints the commands
% make hello         # and avoids unnecessary work
make: `hello' is up to date.
% ./hello
```

Running a Make recipe

```
% make hello
cc -o hello hello.c  # Make prints the commands
% make hello         # and avoids unnecessary work
make: `hello' is up to date.
% ./hello
Hello, CS 211!
%
```

Cleaning up

%

Cleaning up

```
% cd ..              # change to parent directory
```

Cleaning up

```
% cd ..              # change to parent directory
%
```

Cleaning up

```
% cd ..              # change to parent directory
% rm -R cs211        # remove recursively
```

Cleaning up

```
% cd ..              # change to parent directory
% rm -R cs211        # remove recursively
%
```

Cleaning up

```
% cd ..              # change to parent directory
% rm -R cs211        # remove recursively
% mkdir cs211        # make it again
```

Cleaning up

```
% cd ..              # change to parent directory
% rm -R cs211        # remove recursively
% mkdir cs211        # make it again
%
```

Up next

The Edit–Compile–Run Cycle

Compilation

The Unix shell

Building with Make

Getting & building starter code

Appendix: Numeral systems

Appendix: More Make

Getting a Make project onto your shell account

You can unarchive an example Make project in the shell:

```
%
```

Getting a Make project onto your shell account

You can unarchive an example Make project in the shell:

```
% cd cs211
```

Getting a Make project onto your shell account

You can unarchive an example Make project in the shell:

```
% cd cs211
%
```

Getting a Make project onto your shell account

You can unarchive an example Make project in the shell:

```
% cd cs211
% tar -xvf ~cs211/lec/01_compile.tgz
```

Getting a Make project onto your shell account

You can unarchive an example Make project in the shell:

```
% cd cs211
% tar -xvf ~cs211/lec/01_compile.tgz
01_compile/
01_compile/Makefile
01_compile/hello.c
%
```

Getting a Make project onto your shell account

You can unarchive an example Make project in the shell:

```
% cd cs211
% tar -xvf ~cs211/lec/01_compile.tgz
01_compile/
01_compile/Makefile
01_compile/hello.c
% cd 01_compile
```

Getting a Make project onto your shell account

You can unarchive an example Make project in the shell:

```
% cd cs211
% tar -xvf ~cs211/lec/01_compile.tgz
01_compile/
01_compile/Makefile
01_compile/hello.c
% cd 01_compile
%
```

Getting a Make project onto your shell account

You can unarchive an example Make project in the shell:

```
% cd cs211
% tar -xvf ~cs211/lec/01_compile.tgz
01_compile/
01_compile/Makefile
01_compile/hello.c
% cd 01_compile
% ls
```

Getting a Make project onto your shell account

You can unarchive an example Make project in the shell:

```
% cd cs211
% tar -xvf ~cs211/lec/01_compile.tgz
01_compile/
01_compile/Makefile
01_compile/hello.c
% cd 01_compile
% ls
Makefile  hello.c
%
```

– Next time: Imperative programming –

```c
struct page* intentionally_blank = NULL;
```

Up next

Compilation

The Unix shell

Building with Make

Getting & building starter code

Appendix: Numeral systems

Appendix: More Make

Numeral systems

base	counting
2 (binary)	0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011
3 (ternary)	0, 1, 2, 10, 11, 12, 20, 21, 22, 100, 101, 102
5 (quinary)	0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21
8 (octal)	0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13
9 (nonary)	0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12
10 (decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Numeral systems

base	counting
2 (binary)	0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011
3 (ternary)	0, 1, 2, 10, 11, 12, 20, 21, 22, 100, 101, 102
5 (quinary)	0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21
8 (octal)	0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13
9 (nonary)	0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12
10 (decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

base	counting
10 (decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
11 (undecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, 10, 11, 12, 13, 14, 15, 16
12 (duodecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, 10, 11, 12, 13, 14, 15
14 (tetradecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, 10, 11, 12, 13
15 (pentadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, 10, 11, 12
16 (hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11
17 (heptadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, 10

Up next

Compilation

The Unix shell

Building with Make

Getting & building starter code

Appendix: Numeral systems

Appendix: More Make

A fancier `Makefile`

```
%
```

A fancier `Makefile`

```
% cat Makefile
```

A fancier `Makefile`

```
% cat Makefile
# For building CS 211 Lecture 1

CC     ?= cc
CFLAGS  = -std=c11 -pedantic -Wall

all: hello

hello: hello.c
	$(CC) -o $@ $^ $(CFLAGS)

clean:
	rm -f hello

.PHONY: all clean
```

Building the project using Make

%

Building the project using Make

```
% make
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
%
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
Hello, CS 211!
%
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' hello.c
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' hello.c
%
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' hello.c
% ./hello
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' hello.c
% ./hello
Hello, CS 211!
%
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' hello.c
% ./hello
Hello, CS 211!
% make
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' hello.c
% ./hello
Hello, CS 211!
% make
cc -o hello hello.c -std=c11 -pedant…
%
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' hello.c
% ./hello
Hello, CS 211!
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
```

Building the project using Make

```
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' hello.c
% ./hello
Hello, CS 211!
% make
cc -o hello hello.c -std=c11 -pedant…
% ./hello
Hello, everyone!
%
```