The Edit–Compile–Run Cycle

CS 211

Winter 2020
Road map

- Compilation
- Using the shell
- Using Make
- Using starter code
So you’ve written a C program:

```c
#include <stdio.h>

int main()
{
    printf("Hello, CS211!\n");
}
```

What now?
Compilation

We need to translate our program from

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

to

- machine code (machine executable, e.g., x86-64 or ARM).
What does machine code look like? (1/3)

<table>
<thead>
<tr>
<th></th>
<th>85</th>
<th>72</th>
<th>137</th>
<th>229</th>
<th>72</th>
<th>131</th>
<th>236</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>72</td>
<td>141</td>
<td>61</td>
<td>55</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>176</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>232</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>201</td>
</tr>
<tr>
<td>3</td>
<td>137</td>
<td>69</td>
<td>252</td>
<td>137</td>
<td>200</td>
<td>72</td>
<td>131</td>
<td>196</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>93</td>
<td>195</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Each byte value ranges from 0 to 255.)
What does machine code look like? (2/3)

Each byte value ranges from 0x00 to 0xFF.

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>48</td>
<td>89</td>
<td>E5</td>
<td>48</td>
<td>83</td>
<td>EC</td>
<td>10</td>
</tr>
<tr>
<td>48</td>
<td>8D</td>
<td>3D</td>
<td>37</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>B0</td>
</tr>
<tr>
<td>00</td>
<td>E8</td>
<td>0E</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>31</td>
<td>C9</td>
</tr>
<tr>
<td>89</td>
<td>45</td>
<td>FC</td>
<td>89</td>
<td>C8</td>
<td>48</td>
<td>83</td>
<td>C4</td>
</tr>
<tr>
<td>10</td>
<td>5D</td>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What does machine code look like? (2/3)

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

(These numbers are written in base 16, a/k/a hexadecimal, which uses letters A–F for digits greater than 9.)

<table>
<thead>
<tr>
<th>55</th>
<th>48</th>
<th>89</th>
<th>E5</th>
<th>48</th>
<th>83</th>
<th>EC</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>8D</td>
<td>3D</td>
<td>37</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>B0</td>
</tr>
<tr>
<td>00</td>
<td>E8</td>
<td>0E</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>31</td>
</tr>
<tr>
<td>89</td>
<td>45</td>
<td>FC</td>
<td>89</td>
<td>C8</td>
<td>48</td>
<td>83</td>
<td>C4</td>
</tr>
<tr>
<td>10</td>
<td>5D</td>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What does machine code look like? (3/3)

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

(Machine code printed as assembly language mnemonics.)
The Unix shell
Using Unix

For the first few weeks of class, we are going to develop and test our programs under 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; do
  > convert -geometry 128x128 "$i" "thumbs/$i"
> done
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

$ dev

mkdir cs211
cd cs211
emacs -nw hello.c
ls
hello.c
cc hello.c -o hello
ls
hello hello.c
./hello
Hello, CS 211!

You'll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$  dev
%

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
%

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
%

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
% emacs -nw hello.c

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
% emacs -nw hello.c
%

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
% emacs –nw hello.c
% ls

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
% emacs -nw hello.c
% ls
hello.c
%

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
% emacs -nw hello.c
% ls
hello.c
% cc hello.c -o hello

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
% emacs -nw hello.c
% ls
hello.c
% cc hello.c -o hello
%

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
% emacs -nw hello.c
% ls
hello.c
% cc hello.c -o hello
% ls

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
% emacs -nw hello.c
% ls
hello.c
% cc hello.c -o hello
% ls
hello  hello.c
%

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
% emacs -nw hello.c
% ls
hello.c
% cc hello.c -o hello
% ls
hello hello.c
% ./hello

You’ll need to do some setup to enable the dev command…
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
% emacs -nw hello.c
% ls
hello.c
% cc hello.c -o hello
% ls
hello  hello.c
% ./hello
Hello, CS 211!
%

You’ll need to do some setup to enable the dev command…
Building with 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

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

% cc -o hello hello.c
% make hello
make: `hello' is up to date.
% ./hello
Hello, CS 211!
Running a Make recipe

% make hello
Running a Make recipe

% make hello
cc -o hello hello.c
%
Running a Make recipe

% make hello
cc -o hello hello.c
% make hello

Hello, CS 211!
Running a Make recipe

% make hello
cc -o hello hello.c
% make hello
make: `hello' is up to date.
%
Running a Make recipe

% make hello
c  _o hello hello.c
% make hello
make: `hello' is up to date.
% ./hello
Running a Make recipe

% make hello
cc -o hello hello.c
% make hello
make: `hello' is up to date.
% ./hello
Hello, CS 211!
%
Cleaning up

%
Cleaning up

% cd ..
Cleaning up

% cd ..
%
%
Cleaning up

```bash
% cd ..
% rm -Rf cs211
```
Cleaning up

```bash
% cd ..
% rm -Rf cs211
%```


Cleaning up

```bash
% cd ..
% rm -Rf cs211
% mkdir cs211
```
Cleaning up

% cd ..
% rm -Rf cs211
% mkdir cs211
%
Getting & building starter code
Getting a Make project onto eecs

You can download an example Make project from the course website:

%
Getting a Make project onto eecs

You can download an example Make project from the course website:

```
% cd cs211
```
Getting a Make project onto eecs

You can download an example Make project from the course website:

```
% cd cs211
%
%```
Getting a Make project onto eecs

You can download an example Make project from the course website:

```bash
% cd cs211
% wget $URL211/lec/01compile.tgz
```
Getting a Make project onto eecs

You can download an example Make project from the course website:

% cd cs211
% wget $URL211/lec/01compile.tgz
...
%
Getting a Make project onto eeCS

You can download an example Make project from the course website:

```bash
% cd cs211
% wget $URL211/lec/01compile.tgz
...
% tar zxf 01compile.tgz
```
Getting a Make project onto eecs

You can download an example Make project from the course website:

```
% cd cs211
% wget $URL211/lec/01compile.tgz
...
% tar zxf 01compile.tgz
% ```
Getting a Make project onto eecs

You can download an example Make project from the course website:

```
% cd cs211
% wget $URL211/lec/01compile.tgz
...
% tar zxf 01compile.tgz
% cd 01compile
```
Getting a Make project onto eecs

You can download an example Make project from the course website:

```%
cd cs211
wget $URL211/lec/01compile.tgz
...
tar zxf 01compile.tgz
cd 01compile
%```
Getting a Make project onto eecs

You can download an example Make project from the course website:

```bash
% cd cs211
% wget $URL211/lec/01compile.tgz
...
% tar zxf 01compile.tgz
% cd 01compile
% ls
```
Getting a Make project onto eecs

You can download an example Make project from the course website:

```bash
% cd cs211
% wget $URL211/lec/01compile.tgz
...
% tar zxf 01compile.tgz
% cd 01compile
% ls
Makefile src
%```
Getting a Make project onto eecs

You can download an example Make project from the course website:

```bash
% cd cs211
% wget $URL211/lec/01compile.tgz
...
% tar zxf 01compile.tgz
% cd 01compile
% ls
Makefile  src
% ls src
```
Getting a Make project onto eecs

You can download an example Make project from the course website:

```
% cd cs211
% wget $URL211/lec/01compile.tgz
...
% tar zxf 01compile.tgz
% cd 01compile
% ls
Makefile  src
% ls src
hello.c
%
```
A fancier Makefile

%
A fancier Makefile

```bash
% cat Makefile
```

```bash
CFLAGS = -std=c11 -pedantic -Wall
all: build/hello
build/hello: src/hello.c
mkdir -p build
cc -o build/hello src/hello.c $(CFLAGS)
clean:
rm -Rf build
.PHONY: all clean
```
A fancier Makefile

% cat Makefile
# For building CS 211 Lecture 1

CFLAGS = -std=c11 -pedantic -Wall

all: build/hello

build/hello: src/hello.c
    mkdir -p build
    cc -o build/hello src/hello.c $(CFLAGS)

clean:
    rm -Rf build

.PHONY: all clean
%


Building the project using Make

```bash
% make

mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...

build/hello
Hello, CS 211!

sed -i -e 's/CS 211/everyone/' src/hello.c

build/hello
Hello, everyone!

% make

mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...

build/hello
Hello, everyone!
```
Building the project using Make

% make

% make

build/hello
Hello, CS 211!

% sed -i -e 's/CS 211/everyone/' src/hello.c

% make

build/hello
Hello, everyone!
Building the project using Make

% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
%
Building the project using Make

```bash
% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
```
Building the project using Make

% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
Hello, CS 211!
%

Building the project using Make

% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' src/hello.c
Building the project using Make

% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' src/hello.c
%
Building the project using Make

% make
mkdir -p build
c -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' src/hello.c
% build/hello
Building the project using Make

% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' src/hello.c
% build/hello
Hello, CS 211!
%
Building the project using Make

% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' src/hello.c
% build/hello
Hello, CS 211!
% make
Building the project using Make

% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' src/hello.c
% build/hello
Hello, CS 211!
% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
%
Building the project using Make

```bash
% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' src/hello.c
% build/hello
Hello, CS 211!
% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
```
Building the project using Make

% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
Hello, CS 211!
% sed -i -e 's/CS 211/everyone/' src/hello.c
% build/hello
Hello, CS 211!
% make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedant...
% build/hello
Hello, everyone!
%
– Next time: C syntax & more compilation –
## Numeral systems

<table>
<thead>
<tr>
<th>base</th>
<th>counting</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (binary)</td>
<td>0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011</td>
</tr>
<tr>
<td>3 (ternary)</td>
<td>0, 1, 2, 10, 11, 12, 20, 21, 22, 100, 101, 102</td>
</tr>
<tr>
<td>5 (quinary)</td>
<td>0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21</td>
</tr>
<tr>
<td>8 (octal)</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13</td>
</tr>
<tr>
<td>9 (nonary)</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12</td>
</tr>
<tr>
<td>10 (decimal)</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11</td>
</tr>
</tbody>
</table>
## Numeral systems

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

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