The Edit–Compile–Run Cycle

CS 211/Spring 2020
Road map

Compilation

The Unix shell

Building with Make

Getting & building starter code

Appendix: Numeral systems
Compilation
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)

(Each byte value ranges from 0 to 255.)

<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>85</td>
<td>72</td>
<td>137</td>
<td>229</td>
<td>72</td>
<td>131</td>
<td>236</td>
<td>16</td>
</tr>
<tr>
<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>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>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>16</td>
<td>93</td>
<td>195</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What does machine code look like? (2/3)

<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>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>

(Each byte value ranges from 0x00 to 0xFF.)
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 as digits with values 10–15.)
What does machine code look like? (3/3)

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

(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.
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:

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

```bash
$ mkdir thumbs
$ for i in *.png
    do
        convert -geometry 128x128 "$i" "thumbs/$i"
    done
```
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:

```bash
#!/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

# enter development mode

mkdir cs211

# make directory

cd cs211

# change directory

emacs -nw hello.c

# edit (new) file

ls

# list directory contents

cc hello.c

# compile C program

ls

# list directory contents

./a.out

# run compiled program

Hello, CS 211!

(In this week's lab you'll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

$  dev  # enter development mode

(In this week’s lab you’ll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

$ dev # enter development mode
%

(In this week's lab you'll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

$ dev                     # enter development mode
% mkdir cs211             # make directory

(In this week’s lab you’ll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

$ dev
  # enter development mode
%
% mkdir cs211
  # make directory
%

(In this week's lab you'll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

$ dev
# enter development mode
%
mkdir cs211
# make directory
%
cd cs211
# change directory
%
emacs -nw 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 you’ll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

$ dev
% mkdir cs211
% cd cs211
%

(In this week’s lab you’ll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

$ dev
# enter development mode
% mkdir cs211
# make directory
% cd cs211
# change directory
% emacs -nw hello.c
# edit (new) file

(In this week's lab you’ll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

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

(In this week’s lab you’ll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

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

# enter development mode
# make directory
# change directory
# edit (new) file
# list directory contents

(In this week's lab you'll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

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

# enter development mode
# make directory
# change directory
# edit (new) file
# list directory contents

(In this week’s lab you’ll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

$ dev
# enter development mode
% mkdir cs211
# make directory
% cd cs211
# change directory
% emacs -nw hello.c
# edit (new) file
% ls
# list directory contents
hello.c
% cc hello.c
# compile C program

(In this week's lab you'll do the necessary 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
%

# enter development mode
# make directory
# change directory
# edit (new) file
# list directory contents
# compile C program

(In this week's lab you'll do the necessary 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
% ls

(In this week's lab you'll do the necessary setup to enable the dev command.)
Compilation in the Unix shell

$ dev  # enter development mode
% mkdir cs211  # make directory
% cd cs211  # change directory
% emacs -nw 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 you'll do the necessary 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
% ls
  a.out  hello.c
  ./a.out

(In this week's lab you’ll do the necessary 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
% ls
  a.out  hello.c
% ./a.out
Hello, CS 211!
%

(In this week’s lab you’ll do the necessary 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 Maketo automate builds for us.
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

```bash
make hello
# Make prints the commands
make hello
# and avoids unnecessary work
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  # Make prints the commands
%
./hello
Hello, CS 211!
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.
%

./hello
Hello, CS 211!

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
%
Getting & building starter code
Getting a Make project onto eecs

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

```bash
  %
  cd cs211
  %
  curl $LEC211/01compile.tgz | tar zxv
  01compile/
  01compile/Makefile
  01compile/hello.c
  %
  cd 01compile
  %
  ls
    Makefile  hello.c
```
Getting a Make project onto eecs

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

```bash
% 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:

% cd cs211
% curl $LEC211/01compile.tgz | tar zxvk
Getting a Make project onto eecs

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

```
% cd cs211
% curl $LEC211/01compile.tgz | tar zxv
01compile/
01compile/Makefile
01compile/hello.c
%
```
Getting a Make project onto eecs

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

```
% cd cs211
% curl $LEC211/01compile.tgz | tar zxvk
01compile/
01compile/Makefile
01compile/hello.c
% cd 01compile
```
Getting a Make project onto eecs

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

```bash
% cd cs211
% curl $LEC211/01compile.tgz | tar zxvk
01compile/
01compile/Makefile
01compile/hello.c
% cd 01compile
%
```
Getting a Make project onto eecs

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

```bash
% cd cs211
% curl $LEC211/01compile.tgz | tar zxvk
01compile/
01compile/Makefile
01compile/hello.c
% cd 01compile
% ls
```
Getting a Make project onto eecs

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

```bash
% cd cs211
% curl $LEC211/01compile.tgz | tar zxvk
01compile/
01compile/Makefile
01compile/hello.c
% cd 01compile
% ls
Makefile hello.c
%```
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

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

sed -i -e 's/CS 211/everyone/\nhello.c
./hello
Hello, everyone!

make
cc -o hello hello.c -std=c11 -pedant...
./hello
Hello, everyone!
```
Building the project using Make

% make

./hello

Hello, CS 211!

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

./hello

Hello, everyone!

% make

./hello

Hello, everyone!
Building the project using Make

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

Hello, CS 211!

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

Hello, everyone!
Building the project using Make

% make
c c -o hello hello.c -std=c11 -pedant...
% ./hello
Building the project using Make

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

sed -i -e 's/CS 211/everyone/' hello.c
% ./hello
Hello, everyone!
%

22
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
% make
c -o hello hello.c -std=c11 -pedant...
% ./hello
Hello, everyone!
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
Hello, everyone!
Building the project using Make

% make
cmake 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
c c -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
c c -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!
%
- Next time: C syntax -
Appendix: Numeral systems
## 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>
<tr>
<td>10 (decimal)</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11</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, 17</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, 16</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>