# The Edit-Compile-Run Cycle

EECS 211

Winter 2019

So you've written a C program:

```
#include <stdio.h>

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

What now?

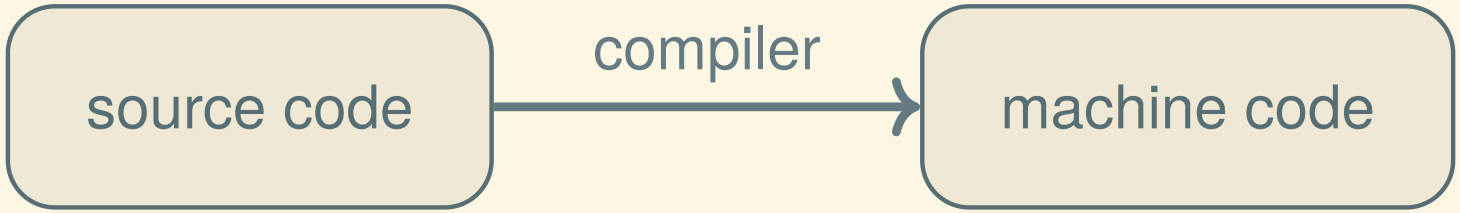
# Compilation

We need to translate our program from

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

to

- machine code (machine executable).



# What does machine code look like?

```
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
```

# What does machine code look like?

```
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
```

# 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 invented 50 years ago. (Modern variants include Linux and Mac OS X.)

# 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 invented 50 years ago. (Modern variants include Linux and Mac OS X.)

**shell** The main program for controlling a Unix computer, using textual 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 invented 50 years ago. (Modern variants include Linux and Mac OS X.) |
| shell | The main program for controlling a Unix computer, using textual commands. |
| terminal | A program (or historically, device) for displaying textual interactions, often remote, with a Unix computer. |

# 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
$ foreach i ( *.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

```
$ exec scl enable devtoolset-6 tcsh
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$ emacs -nw hello.c
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$ emacs -nw hello.c
$
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$ emacs -nw hello.c
$ ls
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$ emacs -nw hello.c
$ ls
hello.c
$
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$ emacs -nw hello.c
$ ls
hello.c
$ cc hello.c -o hello
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$ emacs -nw hello.c
$ ls
hello.c
$ cc hello.c -o hello
$
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$ emacs -nw hello.c
$ ls
hello.c
$ cc hello.c -o hello
$ ls
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$ emacs -nw hello.c
$ ls
hello.c
$ cc hello.c -o hello
$ ls
hello  hello.c
$
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$ emacs -nw hello.c
$ ls
hello.c
$ cc hello.c -o hello
$ ls
hello  hello.c
$ ./hello
```

# Compilation in the Unix shell

```
$ exec scl enable devtoolset-6 tcsh
$ mkdir eecs211
$ cd eecs211
$ emacs -nw hello.c
$ ls
hello.c
$ cc hello.c -o hello
$ ls
hello  hello.c
$ ./hello
Hello, EECS 211!
$
```

# 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.

# Introduction to Make

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.

# Introduction to Make

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
```

# Introduction to Make

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`.*)

# Introduction to Make

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`.*)

Using Make:

```
$
```

# Introduction to Make

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`.*)

Using Make:

```
$ make hello
```

# Introduction to Make

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`.*)

Using Make:

```
$ make hello
cc -o hello helloc
$
```

# Introduction to Make

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`.*)

Using Make:

```
$ make hello
cc -o hello helloc
$ make hello
```

# Introduction to Make

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`.*)

Using Make:

```
$ make hello
cc -o hello helloc
$ make hello
make: `build/hello' is up to date.
$
```

# Cleaning up

$

# Cleaning up

```
$ cd
```

# Cleaning up

```
$ cd
$
```

# Cleaning up

```
$ cd
$ rm -Rf eecs211
```

# Cleaning up

```
$ cd
$ rm -Rf eecs211
$
```

# Cleaning up

```
$ cd
$ rm -Rf eecs211
$ mkdir eecs211
```

# Cleaning up

```
$ cd
$ rm -Rf eecs211
$ mkdir eecs211
$
```

# 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 eecs211
```

# Getting a Make project onto EECS

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

```
$ cd eecs211
$
```

# Getting a Make project onto EECS

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

```
$ cd eecs211
$ wget $URL211/lec/01compile.tgz
```

# Getting a Make project onto EECS

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

```
$ cd eecs211
$ wget $URL211/lec/01compile.tgz
…
$
```

# Getting a Make project onto EECS

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

```
$ cd eecs211
$ 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 eecs211
$ 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 eecs211
$ 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 eecs211
$ 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 eecs211
$ 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:

```
$ cd eecs211
$ 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:

```
$ cd eecs211
$ 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 eecs211
$ wget $URL211/lec/01compile.tgz
…
$ tar zxf 01compile.tgz
$ cd 01compile
$ ls
Makefile  src
$ ls src
hello.c
$
```

# Another `Makefile`

```
$
```

# Another `Makefile`

```
$ cat Makefile
```

# Another `Makefile`

```
$ cat Makefile
CFLAGS = -std=c11 -pedantic -Wall

all: build/hello

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

clean:
	rm -Rf build

.PHONY: all clean
$
```

# Building the project using Make

```
$
```

# Building the project using Make

```
$ make
```

# Building the project using Make

```
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $
```

# Building the project using Make

```
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $ build/hello
```

# Building the project using Make

```
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $ build/hello
  Hello, EECS 211!
  $
```

# Building the project using Make

```
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $ build/hello
  Hello, EECS 211!
  $ sed -i 's/EECS 211/everyone/' src/hello.c
```

# Building the project using Make

```
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $ build/hello
  Hello, EECS 211!
  $ sed -i 's/EECS 211/everyone/' src/hello.c
  $
```

# Building the project using Make

```
$ make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
$ build/hello
Hello, EECS 211!
$ sed -i 's/EECS 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 -pedantic -
Wall
$ build/hello
Hello, EECS 211!
$ sed -i 's/EECS 211/everyone/' src/hello.c
$ build/hello
Hello, EECS 211!
$
```

# Building the project using Make

```
$ make
mkdir -p build
cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
$ build/hello
Hello, EECS 211!
$ sed -i 's/EECS 211/everyone/' src/hello.c
$ build/hello
Hello, EECS 211!
$ make
```

# Building the project using Make

```
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $ build/hello
  Hello, EECS 211!
  $ sed -i 's/EECS 211/everyone/' src/hello.c
  $ build/hello
  Hello, EECS 211!
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $
```

# Building the project using Make

```
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $ build/hello
  Hello, EECS 211!
  $ sed -i 's/EECS 211/everyone/' src/hello.c
  $ build/hello
  Hello, EECS 211!
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $ build/hello
```

# Building the project using Make

```
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $ build/hello
  Hello, EECS 211!
  $ sed -i 's/EECS 211/everyone/' src/hello.c
  $ build/hello
  Hello, EECS 211!
  $ make
  mkdir -p build
  cc -o build/hello src/hello.c -std=c11 -pedantic -
Wall
  $ build/hello
  Hello, everyone!
  $
```