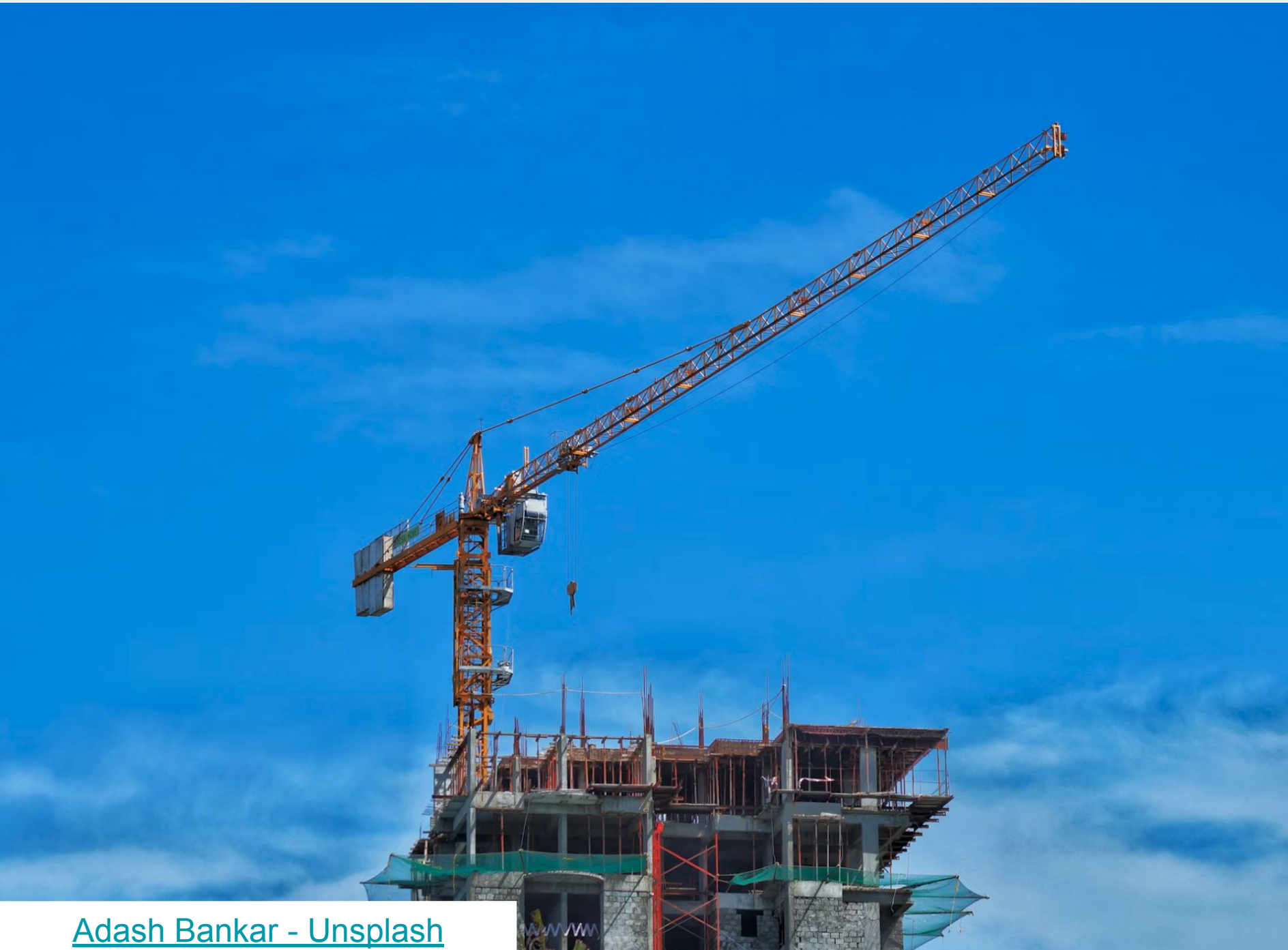
make

Adapted from materials by Dr. Carrier



Adash Bankar - Unsplash

Compiling

How do we compile this program?

Contents of main.c:

```c
#include <stdio.h>

#include "vector.h"
#include "cool_math.h"

int main(){
```

Result of ls:

```
class/c/make_files/example_project$ ls
cool_math.c  cool_math.h  main.c  vector.c  vector.h
```

```
gcc main.c cool_math.c vector.c
```

Compiling

How do we compile this program?

Contents of main.c:

```c
#include <stdio.h>

#include "vector.h"
#include "cool_math.h"

int main(){
```

Result of ls:

```
class/c/make_files/example_project$ ls
cool_math.c  cool_math.h  main.c  vector.c  vector.h
```

```
gcc main.c cool_math.c vector.c
```

Or

```
gcc -c vector.c -o vector.o
gcc -c cool_math.c -o cool_math.o
gcc main.c vector.o cool_math.o
```

How do we remember all that?

We don’t have to! We can use `make`

How do we remember all that?

We don’t have to! We can use `make`

`make` allows us to define rules, which it will recursively follow to compile necessary files

How do we remember all that?

We don’t have to! We can use `make`

`make` allows us to define rules, which it will recursively follow to compile necessary files

We are using GNU make

Running `make`

1. Create a file named Makefile

Running `make`

1. Create a file named Makefile
2. Run `make` (just that one word)
    a. Alternatively you can run `make rule` to run that specific rule

Creating a `Makefile` (no extension)

Makefile is a list of rules.

Creating a `Makefile` (no extension)

Makefile is a list of rules.

Format of a rule:

```
target: prereqs ...
   recipe
   ...
```

Creating a `Makefile` (no extension)

Makefile is a list of rules.

Format of a rule:

```
target: prereqs ...
   recipe
   ...
```

`target` - Name of file to create or action to perform

Creating a `Makefile` (no extension)

Makefile is a list of rules.

Format of a rule:

```
target: prereqs ...
   recipe
   ...
```

`target` - Name of file to create or action to perform

`prereqs` - Files needed to create / perform target

Creating a `Makefile` (no extension)

Makefile is a list of rules.

Format of a rule:

```
target: prereqs ...
   recipe
   ...
```

`target` - Name of file to create or action to perform

`prereqs` - Files needed to create / perform target

`recipe` - Sequence of commands to perform rule

Creating a `Makefile` (no extension)

Makefile is a list of rules.
Format of a rule:

```
target: prereqs ...
   recipe
   ...
```

`target` - Name of file to create or action to perform

`prereqs` - Files needed to create / perform target

`recipe` - Sequence of commands to perform rule

Note, recipes line start with a tab, not spaces!
Ctrl + V then Tab in Vim (if you use spaces via expandtab)

Example (simple)

To compile our example from earlier

```
default: main.c vector.c cool_math.c
   gcc -o program main.c vector.c cool_math.c
```

Example (simple)

To compile our example from earlier

```
default: main.c vector.c cool_math.c
   gcc -o program main.c vector.c cool_math.c
```

Note that running `make` will run the “default” rule, or the first in the file if default doesn’t exist

Example (advanced)

To compile our example from earlier

```
default: program
program: main.c vector.o cool_math.o
   gcc -o program main.c vector.o cool_math.o
vector.o: vector.c
   gcc -o vector.o -c vector.c
cool_math.o: cool_math.c
   gcc -o cool_math.o -c cool_math.c
```

Future compilation will only recompile what's needed! (uses file modified dates/times)

Making changes easier

What if we want to change compilers later?

We don’t want to edit every line…

Making changes easier

What if we want to change compilers later?

We don’t want to edit every line…

We can use variables!

Making changes easier

What if we want to change compilers later?

We don’t want to edit every line…

We can use variables!

```
CC = gcc
CFLAGS = -Wall
program: main.c vector.o
   $(CC) $(CLFLAGS) -o program main.c vector.o
```

Making changes easier

What if we want to change compilers later?

We don’t want to edit every line…

We can use variables!

```
CC = gcc
CFLAGS = -Wall
program: main.c vector.o
   $(CC) $(CLFLAGS) -o program main.c vector.o
```

Note we access variables with $(var)

This is different from bash scripts!

Special tokens

Make gives us some handy tools:

Special tokens

Make gives us some handy tools:

$@ - expands to the name of the current rule

Special tokens

Make gives us some handy tools:

$@ - expands to the name of the current rule

$< - expands to name of first prereq

Special tokens

Make gives us some handy tools:

$@ - expands to the name of the current rule

$< - expands to name of first prereq

$^ - expands to space-separated prereq list

Special tokens

Make gives us some handy tools:

$@ - expands to the name of the current rule

$< - expands to name of first prereq

$^ - expands to space-separated prereq list

% - Matches a pattern that we can use later in line

Special tokens

Make gives us some handy tools:

$@ - expands to the name of the current rule

$< - expands to name of first prereq

$^ - expands to space-separated prereq list

% - Matches a pattern that we can use later in line

```
%.o: %.c
   gcc -o $@ -c $<
```

This generalizes our vector.o and cool_math.o from our example!

Comments

Comments start with #, just like in bash!

Action rules

Some rules don’t create a file, they perform an action!

Action rules

Some rules don’t create a file, they perform an action!

The classic example is clean:

Action rules

Some rules don’t create a file, they perform an action!

The classic example is clean:

```
clean:
   rm program vector.o cool_math.o
```

We can run this with `make clean`

Action rules

Some rules don’t create a file, they perform an action!

The classic example is clean:

```
clean:
   rm program vector.o cool_math.o
```

We can run this with `make clean`

Another common example is the `install` rule

Why `make`?

Why `make`?

Allows us (and others who use our code) to easily compile

Why `make`?

Allows us (and others who use our code) to easily compile

Why use make over a bash script?

Why `make`?

Allows us (and others who use our code) to easily compile

Why use make over a bash script?

- The recursive prereq lookup
- The date-checking to prevent us from recompiling files that haven't changed