Compilation and Makefiles

- Compilers
- Make/Makefiles

❖ Compilers

Compilers are programs that

- convert program source code to executable form
- "**executable**" might be machine code or bytecode

The Gnu C compiler (`gcc`)

- applies source-to-source transformation (pre-processor)
- compiles source code to produce object files
- links object files and libraries to produce executables

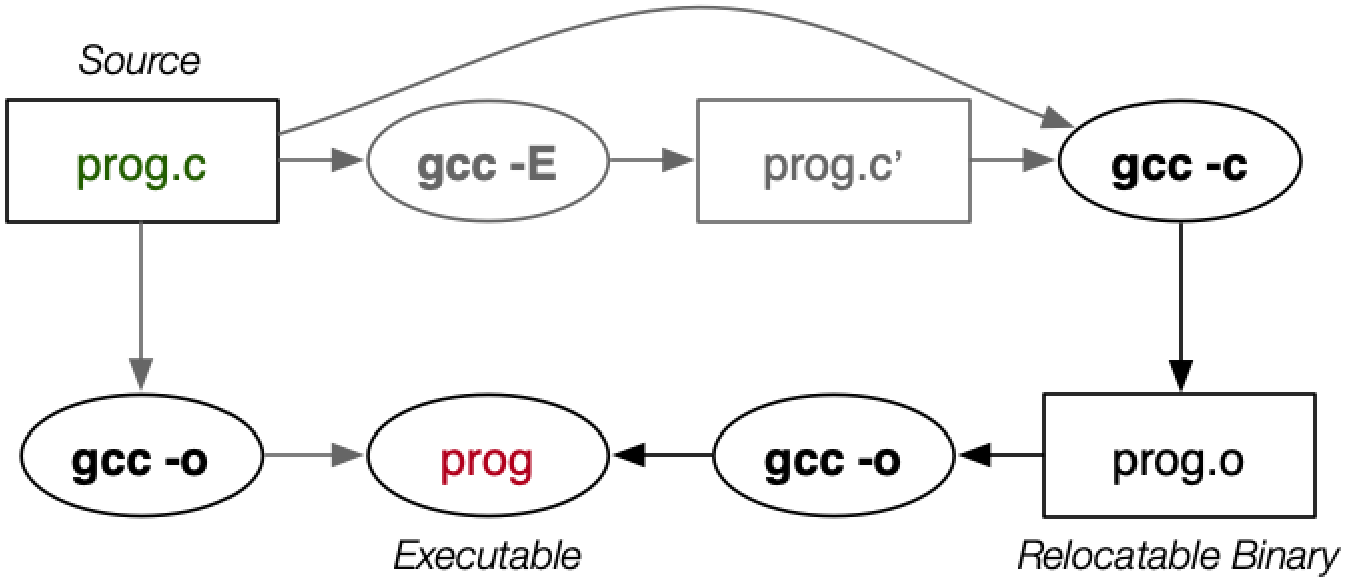
`clang` is an alternative C compiler (also available in CSE)

Note that `dcc` and `3c` are wrappers around `gcc`/`clang`

- providing more checking and more detailed/understandable error messages
- better run-time support (e.g. array bounds, use of dynamic memory)

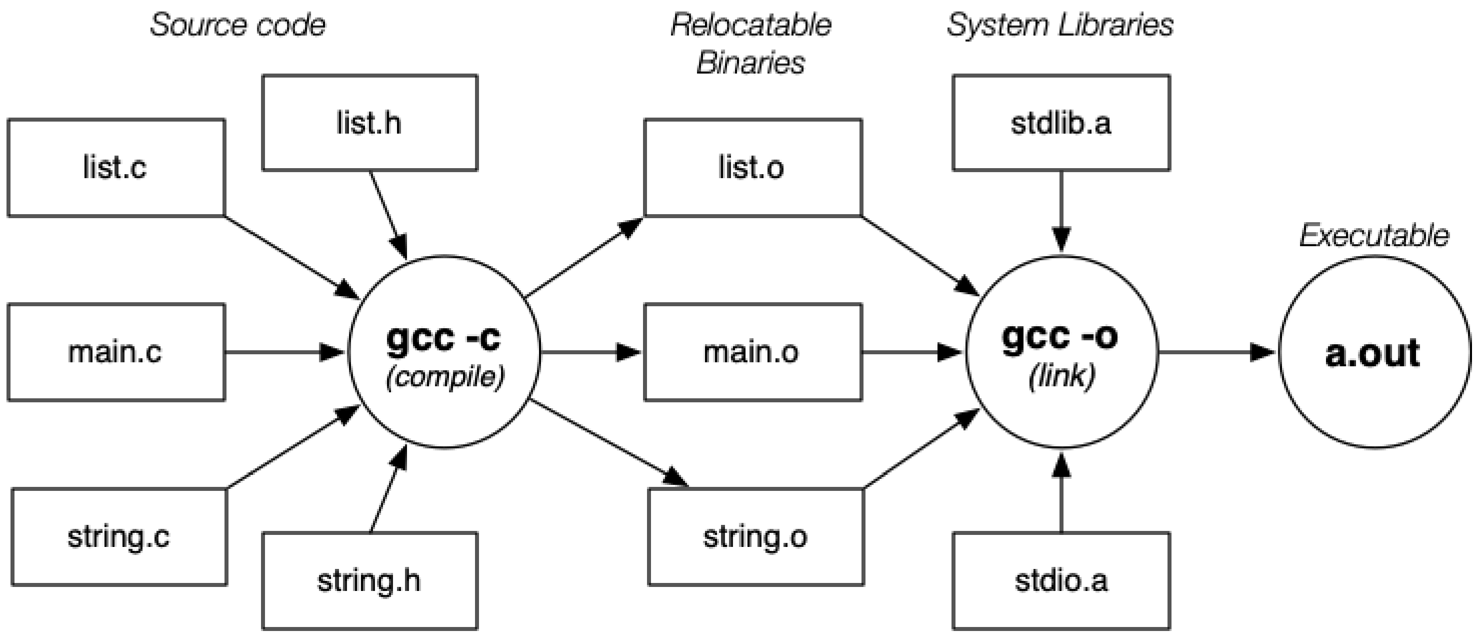
❖ ... Compilers

Stages in C compilation: pre-processing, compilation, linking



❖ ... Compilers

When multiple C files are involved:



❖ ... Compilers

Compilation/linking with **gcc**

```
gcc -c Stack.c
produces Stack.o, from Stack.c and Stack.h
gcc -c bracket.c
produces bracket.o, from bracket.c and Stack.h
gcc -o rbt bracket.o Stack.o
links bracket.o, Stack.o and libraries
producing executable program called rbt
```

Note that **`stdio,assert`** included implicitly.

`gcc` is a multi-purpose tool

- compiles (**`-c`**), links, makes executables (**`-o`**)

❖ Make/Makefiles

Compilation process is complex for large systems.

How much to compile?

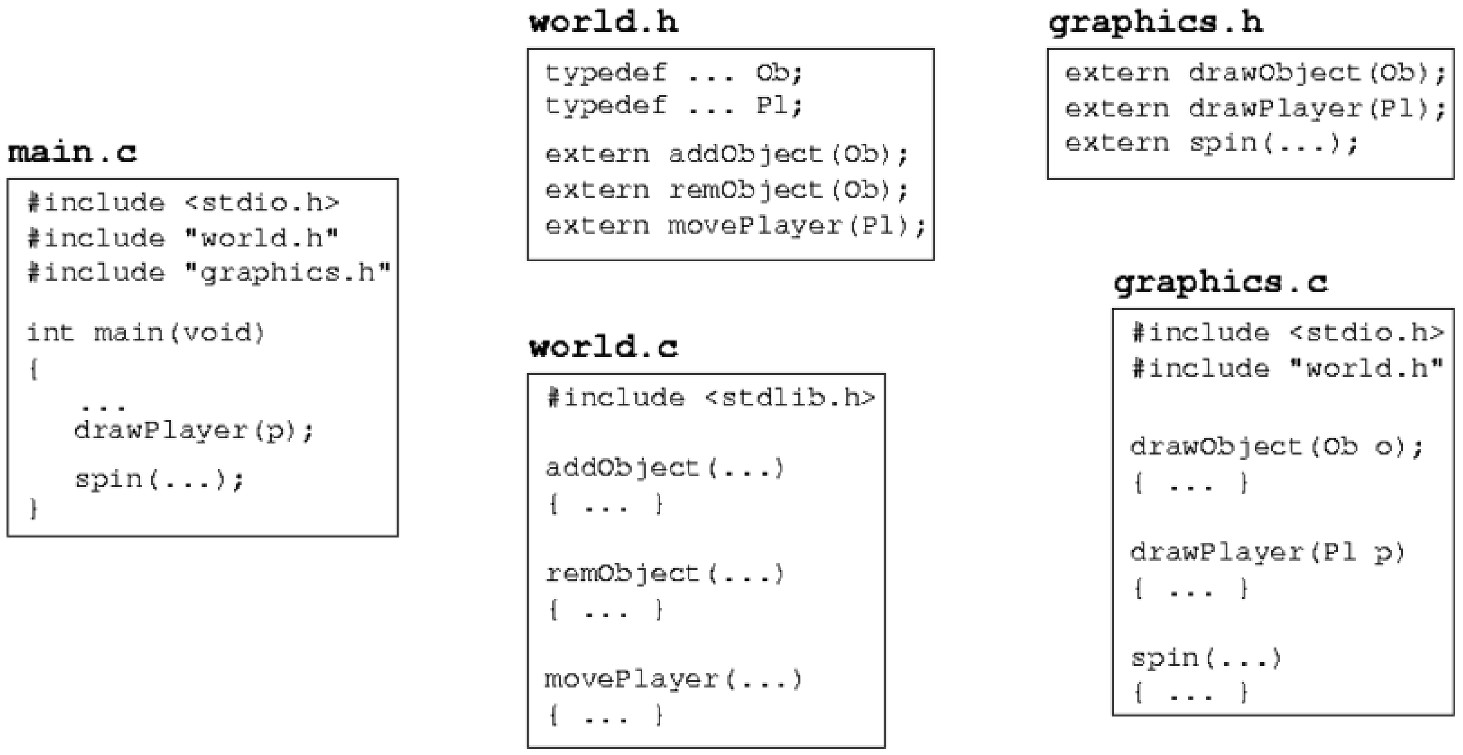
- ideally, what's changed since last compile
- practically, recompile everything, to be sure

The `make` command assists by allowing

- programmers to document dependencies in code
- minimal re-compilation, based on dependencies

❖ ... Make/Makefiles

Example multi-module program ...



main.c

```
#include <stdio.h>
#include "world.h"
#include "graphics.h"

int main(void)
{
   ...
   drawPlayer(p);

   spin(...);
}
```

world.h

```
typedef ... Ob;
typedef ... Pl;

extern addObject(Ob);
extern remObject(Ob);
extern movePlayer(Pl);
```

world.c

```
#include <stdlib.h>

addObject(...)
{ ... }

remObject(...)
{ ... }

movePlayer(...)
{ ... }
```

graphics.h

```
extern drawObject(Ob);
extern drawPlayer(Pl);
extern spin(...);
```

graphics.c

```
#include <stdio.h>
#include "world.h"

drawObject(Ob o);
{ ... }

drawPlayer(Pl p)
{ ... }

spin(...)
{ ... }
```

❖ ... Make/Makefiles

`make` is driven by *dependencies* given in a `Makefile`

A dependency specifies

```
target : source₁ source₂ …
        commands to build target from sources
```

e.g.

```
game : main.o graphics.o world.o
        gcc -o game main.o graphics.o world.o
```

Rule: *target* is rebuilt if older than any $source_i$ (applied recursively)

❖ ... Make/Makefiles

```
game : main.o  graphics.o  world.o
        gcc -o game main.o graphics.o world.o

main.o : main.c graphics.h world.h
        gcc -Wall -Werror -c main.c

graphics.o : graphics.c world.h
        gcc -Wall -Werror -c graphics.c

world.o : world.c
        gcc -Wall -Werror -c world.c
```

Things to note:

- A target (**`game`**, **`main.o`**, ...) is on a newline
  - followed by a **`:`**
  - then followed by the files that the target is dependent on
- The action (**`gcc`** ...) is always on a newline
  - and must be indented with a TAB

❖ ... Make/Makefiles

If **`make`** arguments are targets, build just those targets:

```
prompt$ make world.o
gcc -Wall -Werror -c world.c
```

If no args, build first target in the **`Makefile`**.

```
prompt$ make
gcc -Wall -Werror -c main.c
gcc -Wall -Werror -c graphics.c
gcc -Wall -Werror -c world.c
gcc -o game main.o graphics.o world.o
```

❖ ... Make/Makefiles

`Makefiles` can contain "variables"

- e.g. `CC`, `CFLAGS`, `LDFLAGS`
- can easily change which C compiler used, etc

`make` has rules, which allow it to interpret e.g.

```
Stack.o : Stack.c Stack.h
```

as

```
Stack.o : Stack.c Stack.h
        $(CC) $(CFLAGS) -c Stack.c
```

Produced: 6 Jun 2020