COMP26020 Programming Languages and Paradigms -- Part 1

Modular Compilation

Modular Compilation

- Large programs: need to organise the code in several source files
  - Redis's `src` folder
  - Linux kernel sources (56000 C source files for v6.4!)
- **How to break up the program code into different files representing well isolated compilation units**?
  - Modular compilation, covered in this slide deck

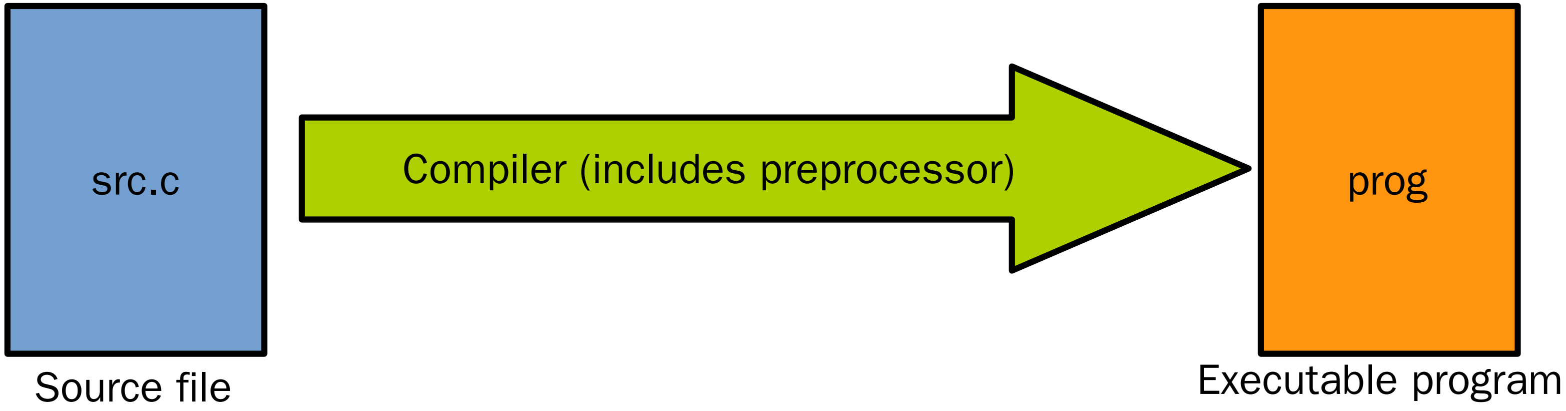
Modular Compilation

- Large programs: need to organise the code in several source files
  - Redis's `src` folder
  - Linux kernel sources (56000 C source files for v6.4!)
- **How to break up the program code into different files representing well isolated compilation units**?
  - Modular compilation, covered in this slide deck
- Also, can't run `gcc <56000 source files>` each time we do a small update to one or a few source files
- **How to automate efficiently the build process?**
  - Automated compilation, covered in the next slide deck

A Closer Look at the Compilation Phase

A Closer Look at the Compilation Phase

```
gcc src.c -o prog
```



src.c

Compiler (includes preprocessor)

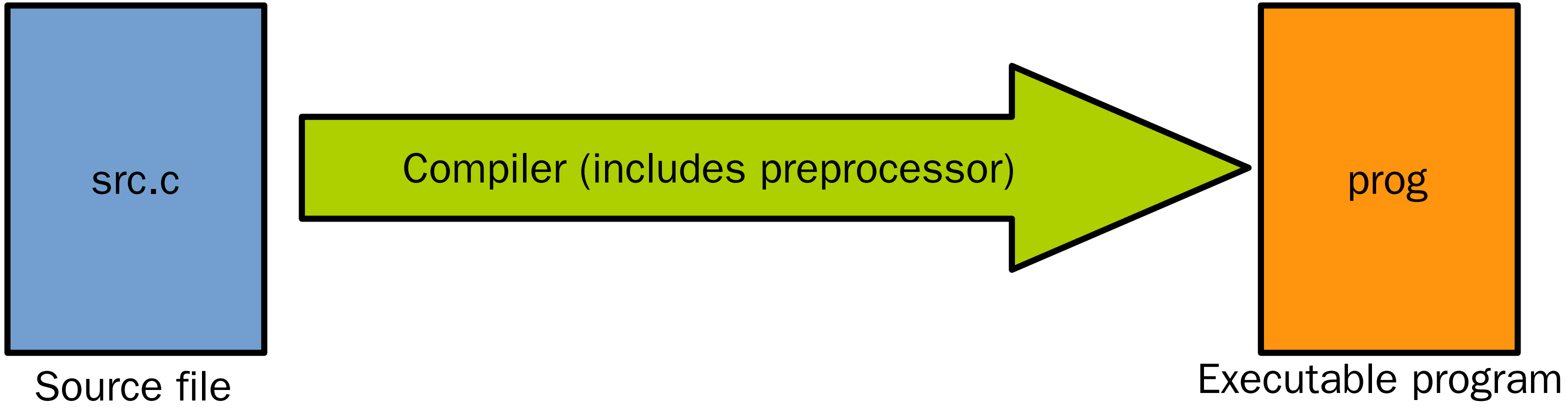
prog

Source file

Executable program

A Closer Look at the Compilation Phase

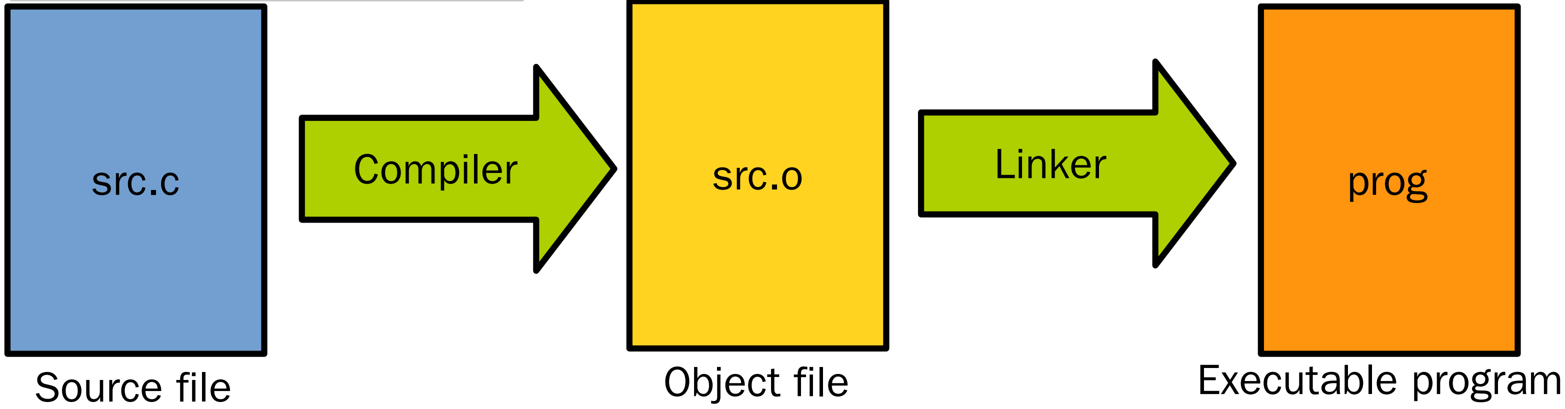
```
gcc src.c -o prog
```



Actually things are still a little bit more complicated 😛

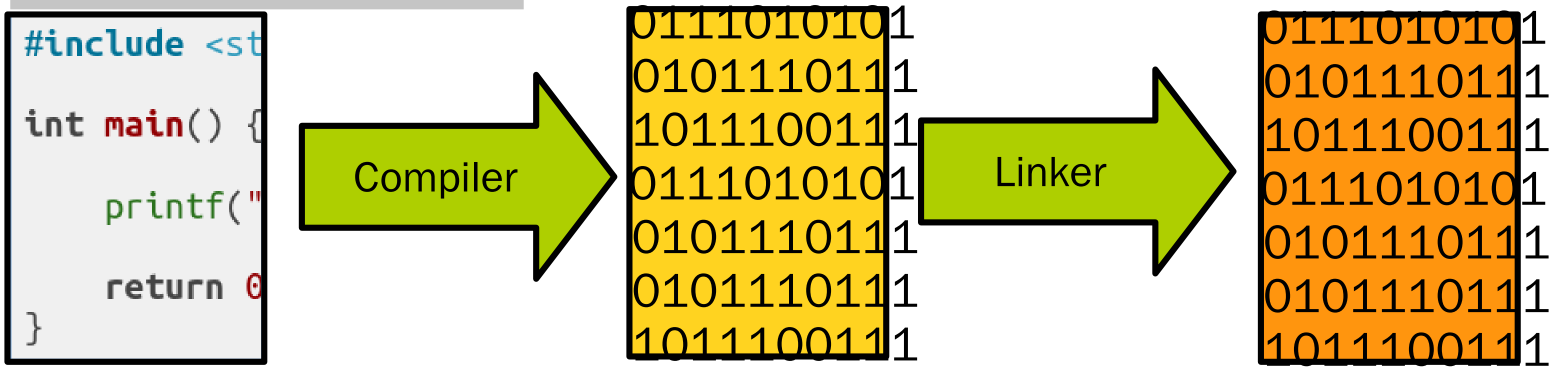
A Closer Look at the Compilation Phase

```
gcc -c src.c -o src.o
gcc src.o -o prog
```



A Closer Look at the Compilation Phase

```
gcc -c src.c -o src.o
gcc src.o -o prog
```



Source file
(text)

Object file
(binary, non-executable)

Executable program
(binary, executable)

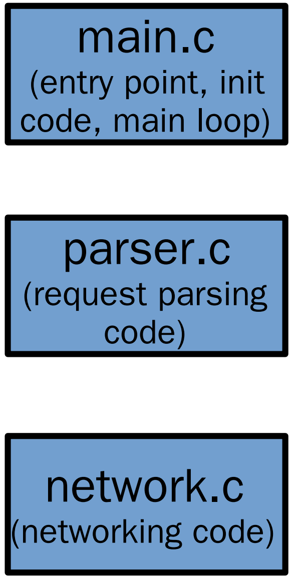
Breaking Down the Program into Modules

Breaking Down the Program into Modules

- Hypothetical example of a server application implemented into 3 `.c` source files, also named **modules**:

Breaking Down the Program into Modules

- Hypothetical example of a server application implemented into 3 `.c` source files, also named **modules**:



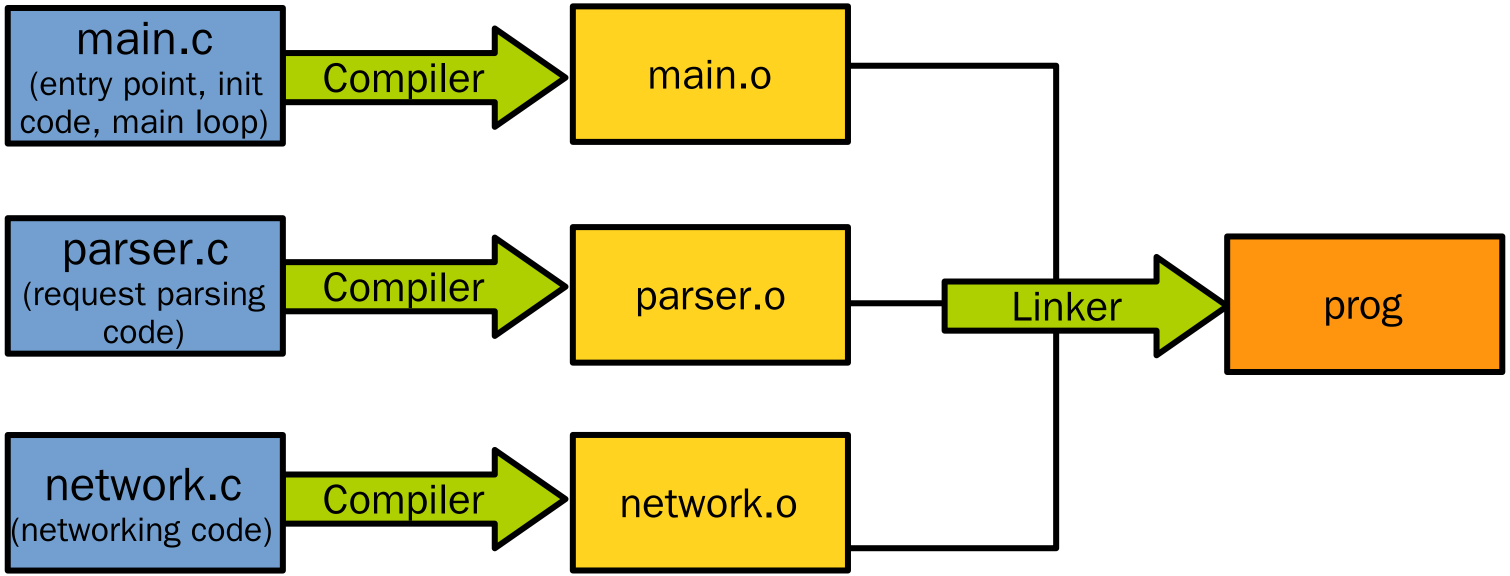
main.c
(entry point, init code, main loop)

parser.c
(request parsing code)

network.c
(networking code)

Breaking Down the Program into Modules

- Hypothetical example of a server application implemented into 3 `.c` source files, also named **modules**:



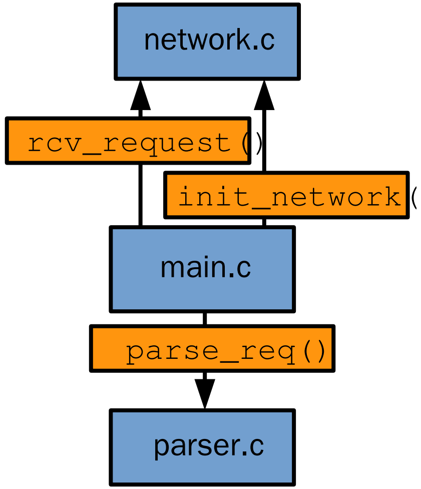
```
gcc -c main.c -o main.o                     # build main.o
gcc -c parser.c -o parser.o                 # build parser.o
gcc -c network.c -o network.o               # build network.o
gcc main.o network.o parser.o -o prog       # link prog
```

Breaking Down the Program into Modules

- Each module (`.c` source file) exposes an **interface** callable from the other source files
    - Should be **as small as possible** to hide internal code/data

Breaking Down the Program into Modules

- Each module (`.c` source file) exposes an **interface** callable from the other source files
  - Should be **as small as possible** to hide internal code/data



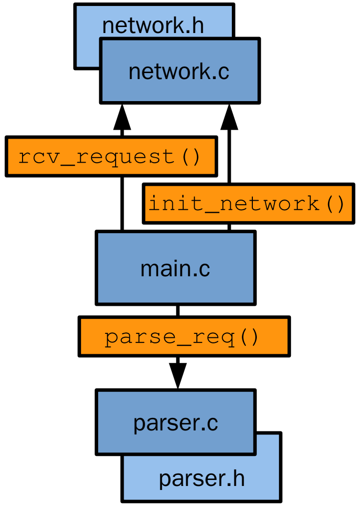
- Assume the following interactions:
  - Srever starts, `main.c` calls `init_network()` implemented in `network.c` to init. networking
  - `main.c` then runs the main sever loop:
    - Calls `rcv_request()` implemented in `network.c` to wait for the next request
    - When received, calls `parse_req()` implemented by `parser.c` to process the request
    - Rince and repeat

Header File for Modular Compilation

- There is generally 1 header file per module, **defining its interface**.
- Can be included in several `.c` files so **must contain only declarations**
    - Function prototypes, struct/typedefs/enums declarations, variable declarations
    - **No definitions** (i.e. function body/global variable initialisation)
- For example `network.h`:

```c
#ifndef NETWORK_H // include guard
#define NETWORK_H

typedef struct {
    int id;
    char content[128];
} request;

void init_network();
int rcv_request(request *r);

#endif /* NETWORK_H */
//
```

16-modular-compilation/network.h

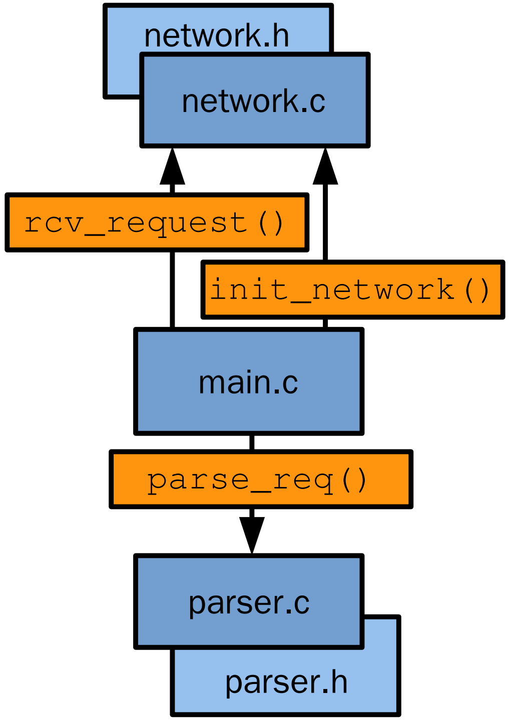


Breaking Down the Program into Modules

- `network.c`:

```
/* std includes here */
#include "network.h"

// this function and variable are internal
// so they are not declared in network.h
// the keyword static force their use
// to be only within the network.c file
static void generate_request(request *r);
static int request_counter = 0;

void init_network() {
    /* init code here ... */
}

int rcv_request(request *r) {
    generate_request(r);
    /* ... */
}

static void generate_request(request *r) {
    /* ... */
}
```

16-modular-compilation/network.c

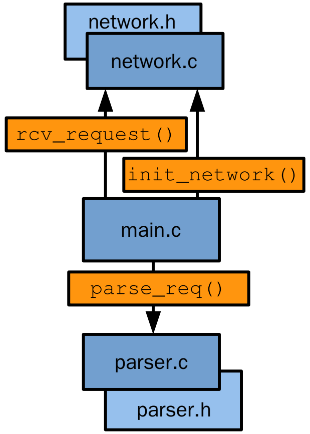


Breaking Down the Program into Modules

- `parser.h`:

```c
#ifndef PARSER_H
#define PARSER_H

/* needed for the definition of request: */
#include "network.h"

void parse_req(request *r);

#endif
```

16-modular-compilation/parser.h



- `parser.c`:

```c
#include <stdio.h>

#include "parser.h"

static void internal1(request *r);
static void internal2(request *r);

void parse_req(request *r) {
    internal1(r);
    internal2(r);
}

static void internal1(request *r) {
    /* ... */
}

static void internal2(request *r) {
    /* ... */
}
```

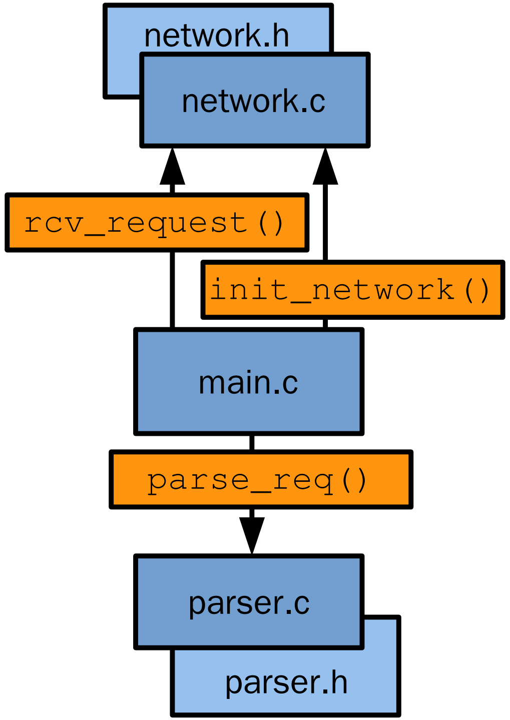
16-modular-compilation/parser.c

Breaking Down the Program into Modules

- `main.c`:

```
#include "network.h"
#include "parser.h"

int main(int argc, char **argv) {
    request req;

    /* call functions from network module */
    init_network();
    rcv_request(&req);

    /* call function from parser module */
    parse_req(&req);

    /* ... */
}
```

[16-modular-compilation/main.c]



Compile & Test

```
# Compile the .c source files, in no particular order:
$ gcc -c main.c -o main.o
$ gcc -c parser.c -o parser.o
$ gcc -c network.c -o network.o

# Link the final executable:
$ gcc main.o parser.o network.o -o prog

# Launch it
$ ./prog
```

Summary

- **Modular Compilation:** breaking down a program's sources into multiple header `.h` files and source `.c` files
    - Proper source organisation is important for medium/large programs
    - Export using header files only the interface supposed to be used from outside the module

---

Feedback form: https://bit.ly/3lInZ8h