Why You Need To Know About Makefile

Reporter :
LIAO CHAO HSIANG

Outline

- What is Makefile ?
- How to use it ?
- Makefile rule
- Example
- Pros
- QA

Makefile Introduction

The make utility is a software tool for managing and maintaining computer programs consisting many component files.

The make utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them.

- GNU Make

  -> For GNU/Linux

- BSD Make

  -> For FreeBSD , OpenBSD

- Microsoft nMake

  -> For Windows , Visual Studio

Makefile contains

- Dependency rules
  - Targets : dependencies
- Suffix(Implicit rules)
  - .SUFFIXES : .foo .bar
- Macros
  - CC := gcc

Usage

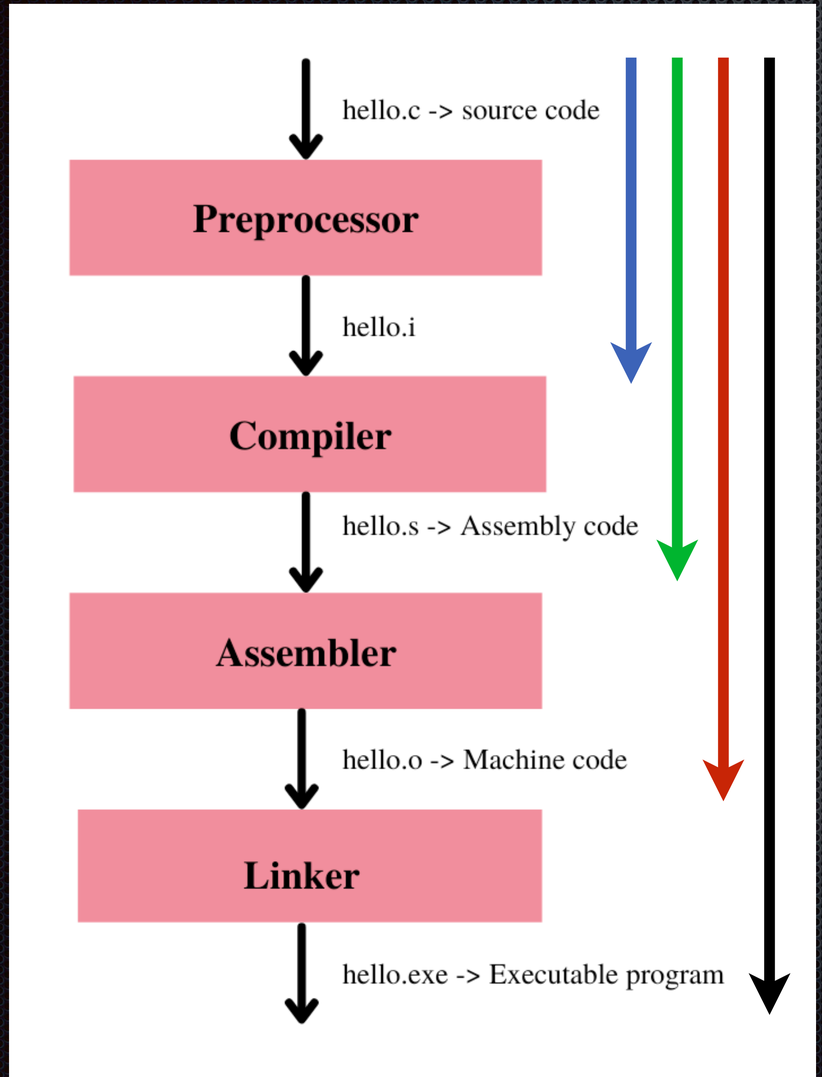
I. Create “Makefile” under project directory

II. Write the rules and shell you need

III. Type “make” in your terminal for project construction

IV. Type “make clean” in your terminal for project reconstruction

GCC command(compiler part)



GCC/Clang

- -E Preprocessor code *.i
- -S Assembly code. *.s
- -C Object file. *.o
- Null Executable file *.exe

GCC command(file part 1)

GCC/Clang

- -o Name for the target
- -I For Include file
- -Wall With warning
- -w Without warning

GCC/Clang

- -static For static library
- -share For dynamic library
- Wildcard For specific pattern
- Patsubst For specific pattern

GCC command(file part 2)

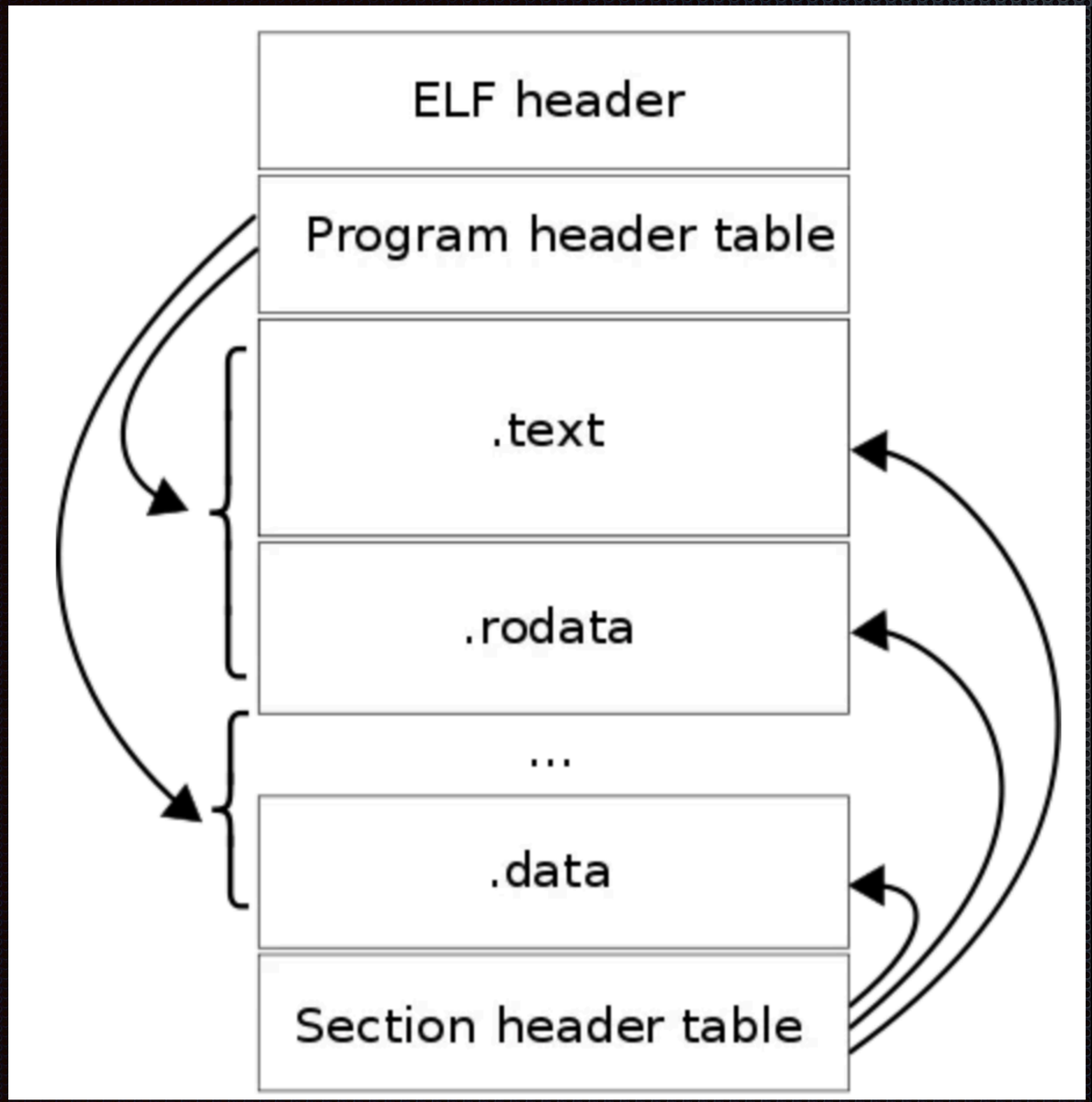
$(wildcard pattern)

$(patsubst pattern, replacement, text)

```
SRC=${wildcard *.c}
#OBJ=main.o funA.o funB.o
#OBJ=${SRC:.c=.o}
OBJ = $(patsubst %.c,$(B_DIR)/%.o,$(SRC))
```

Elf(concise)

Executable and Linkable Format




```
❯ objdump -all-headers ../exe/main

../exe/main:    file format mach-o arm64

Sections:
Idx Name          Size     VMA              Type
  0 __text        00000570 00000001000037f0 TEXT
  1 __stubs       00000054 0000000100003d60 TEXT
  2 __cstring     000001e1 0000000100003db4 DATA
  3 __const       00000008 0000000100003f98 DATA
  4 __unwind_info 0000005c 0000000100003fa0 DATA
  5 __got         00000038 0000000100004000 DATA
  6 __data        00000158 0000000100008000 DATA
  7 __common      00000008 0000000100008158 BSS

Disassembly of section __TEXT,__text:

00000001000037f0 <_set_account_base>:
; set_account_base():
1000037f0: ff c3 00 d1  sub     sp, sp, #48
1000037f4: fd 7b 02 a9  stp     x29, x30, [sp, #32]
1000037f8: fd 83 00 91  add     x29, sp, #32
1000037fc: 28 00 00 b0  adrp    x8, 0x100008000 <_set_account_base+0x20>
100003800: e8 07 00 f9  str     x8, [sp, #8]
100003804: 28 00 00 b0  adrp    x8, 0x100008000 <_set_account_base+0x28>
100003808: 08 01 00 91  add     x8, x8, #0
10000380c: 00 21 00 91  add     x0, x8, #8
100003810: e0 0b 00 f9  str     x0, [sp, #16]
100003814: a0 c3 1f bc  stur    s0, [x29, #-4]
100003818: 55 01 00 94  bl      0x100003d6c <_time+0x100003d6c>
10000381c: e8 07 40 f9  ldr     x8, [sp, #8]
100003820: e0 0b 40 f9  ldr     x0, [sp, #16]
100003824: a0 c3 5f bc  ldur    s0, [x29, #-4]
100003828: 00 01 00 bd  str     s0, [x8]
10000382c: 53 01 00 94  bl      0x100003d78 <_time+0x100003d78>
100003830: fd 7b 42 a9  ldp     x29, x30, [sp, #32]
100003834: ff c3 00 91  add     sp, sp, #48
100003838: c0 03 5f d6  ret
```

Executable file

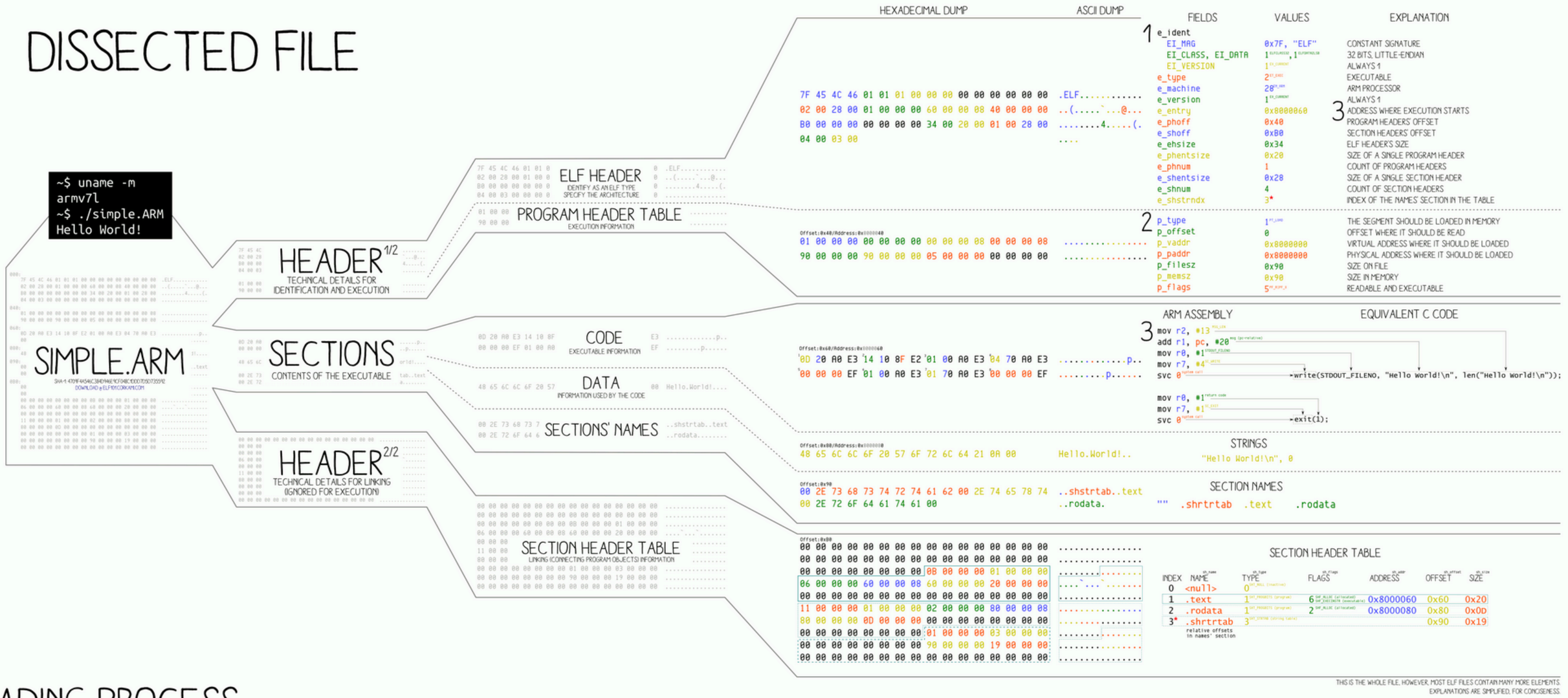
Shared library

Object code

Core dump

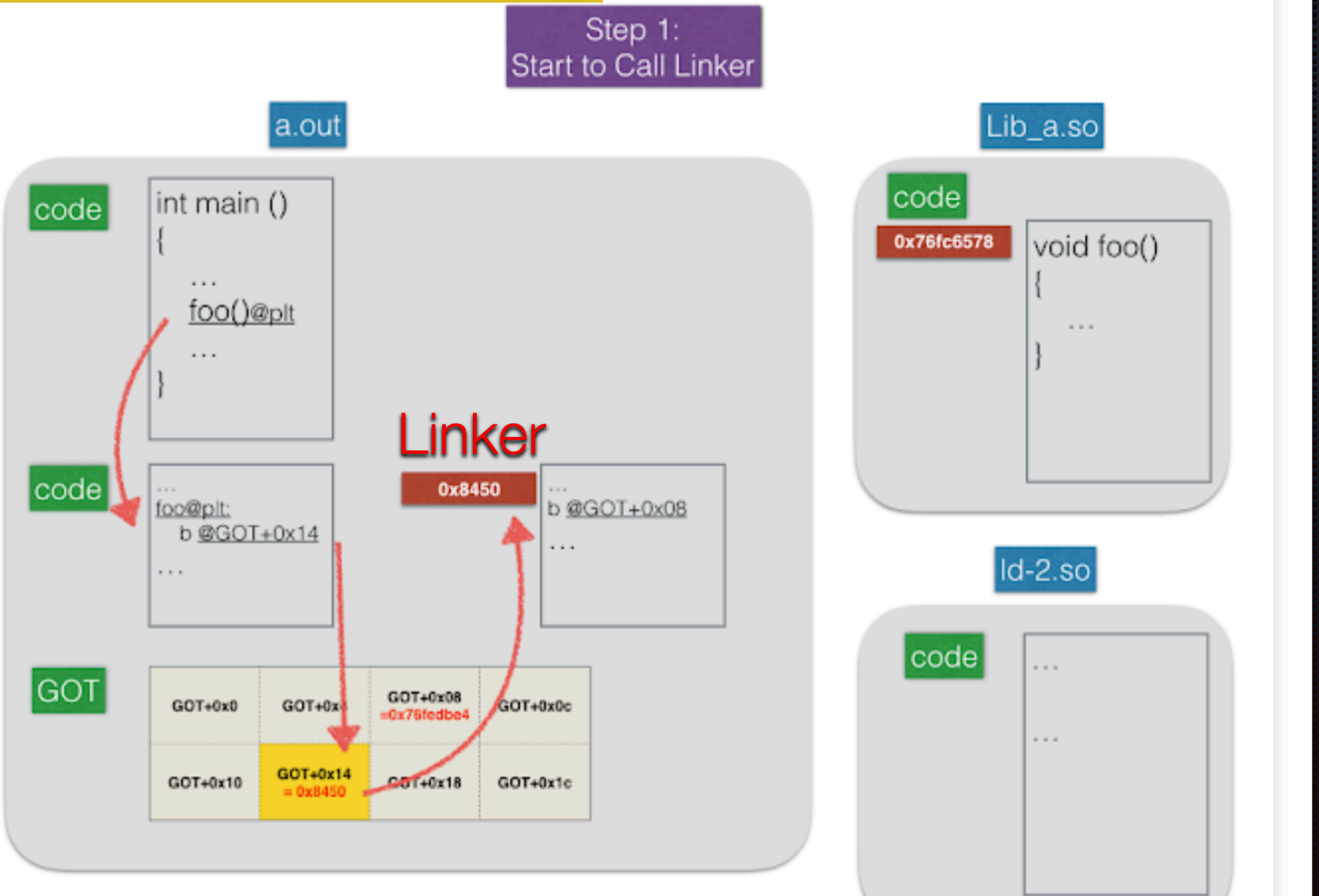
Elf(exhaustive)

DISSECTED FILE

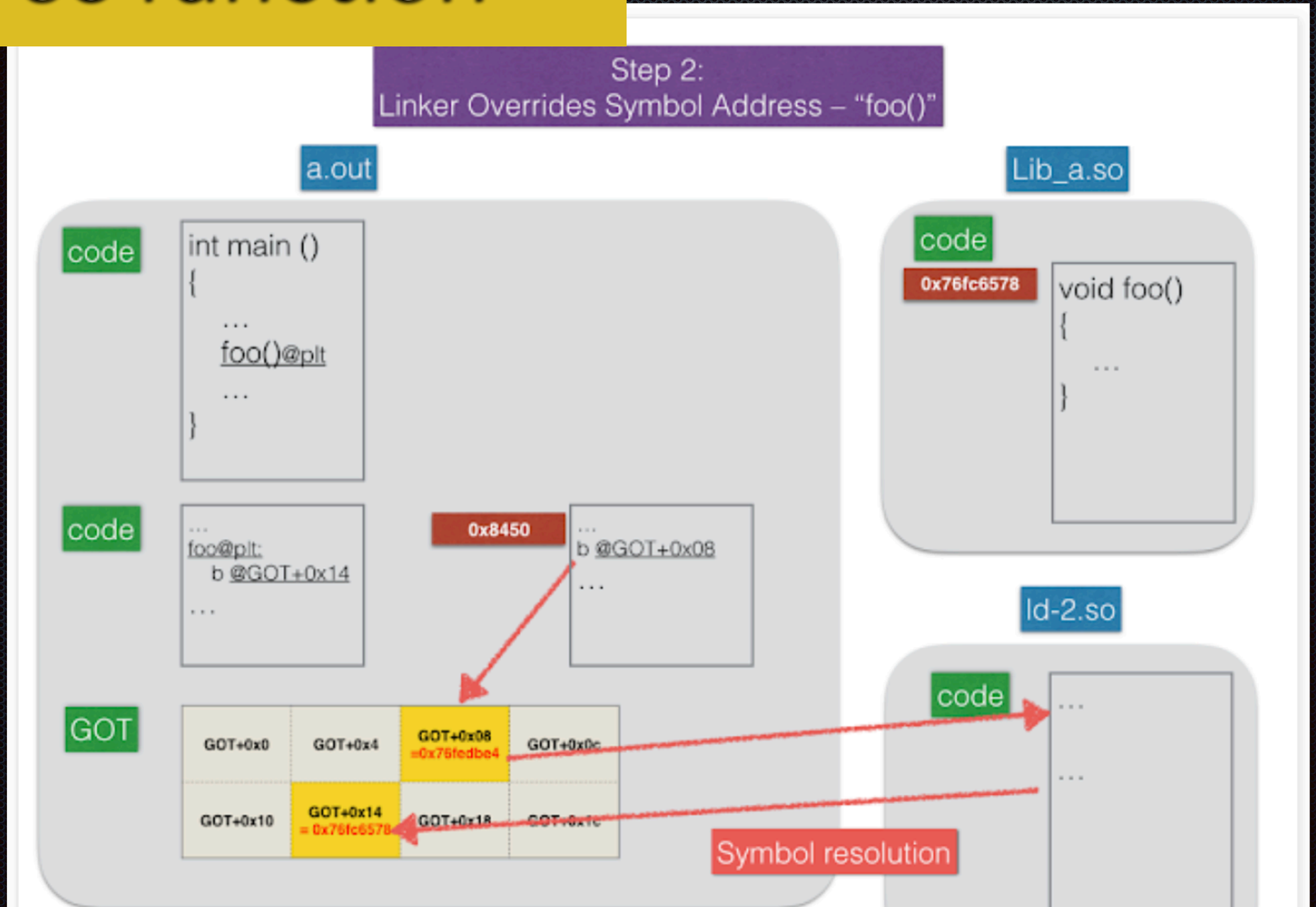


LOADING PROCESS

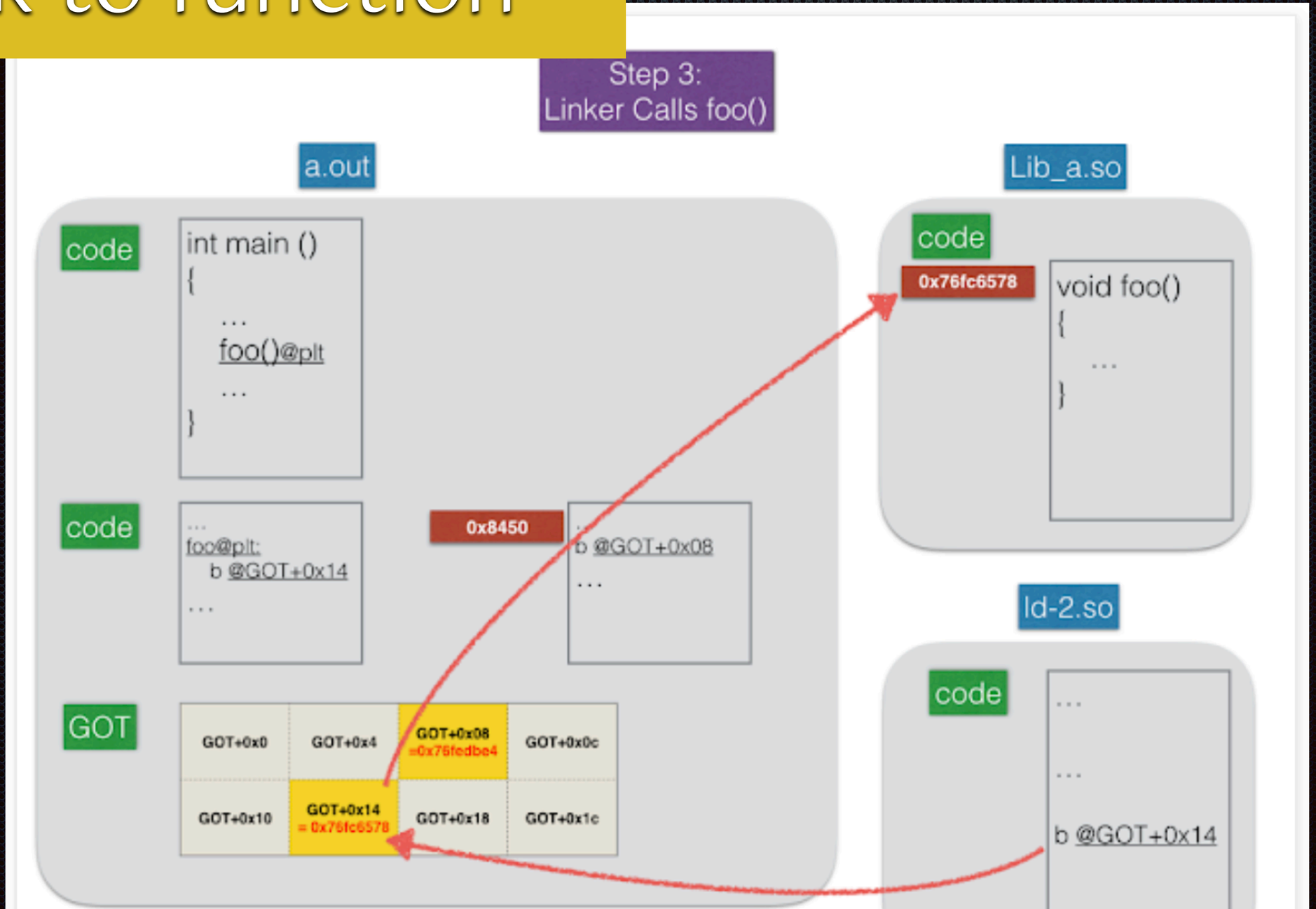
Call linker


Step 1:
Start to Call Linker
a.out
code
int main ()
{
…
foo()@plt
…
}
code
…
foo@plt:
b @GOT+0x14
…
Linker
0x8450
…
b @GOT+0x08
…
GOT
GOT+0x0
GOT+0x08
=0x76fedbe4
GOT+0x0c
GOT+0x10
GOT+0x14
= 0x8450
GOT+0x18
GOT+0x1c
Lib_a.so
code
0x76fc6578
void foo()
{
…
}
ld-2.so
code
…
…

Parse function



Link to function



Dependency Rules

Targets : dependencies

Commands…

Targets for what you want.

Dependencies for what you need.

Commands for what will be done.

*Only Tab is accepted !

```
hello: main.cpp hello.cpp factorial.cpp
    $(CC) $(CFLAGS) $? $(LDFLAGS) -o $@
```

```
main: $(OBJ)
        ${CC} $(OBJ) $(HEAD) -I include  -o $@
        @echo "\nPhase2 -> Create executable file and clean includes\n"
        @sleep 0.5
        mv ${OBJ} ./../build/
        mv ${exe} ./../exe
#       rm ./*.h
```

```
.PHONY: clean
clean:
        @echo "\nClean up objective files and prog....\n"
        @sleep 1
        @rm -rf exe
        @rm -rf build
        @echo "\nDone.\n"
        @tree .
#       @cd src && $(MAKE) clean
#       @rm -rf *.o
```

Suffix (Implicit Rules)

.SUFFIXES : Dependency Target

Dependency Target :

Commands…

Targets for what you want.

Dependencies for what you need.

Commands for what will be done.

*Only Tab is accepted !

Try It

.SUFFIXES : .foo .bar

.foo .bar :

gcc -c *.foo

Result → *.bar

Macros

Variable := “Text”

Variable := $(shell command)

Variable += “Text”

Variable for name of declaration .

Text for what the variable indicate.

Shell commands can also be done.

+= for expand the Text in variable.

CFLAGS := -Wall

CFLAGS += -c

CFLAGS is now “ -Wall -c”

Example

main.c

```c
/*main.c*/
#include<stdio.h>
#include<stdlib.h>
float _main_base;
/*init*/
void funA();
void funB();
void funC();

int main(){

        funA();

        funB();

        funC();


return 0;
}
```

Makefile

```
# -*-Sub  Makefile  -*-
CC:=gcc
INC_DIR := ../include
CFLAGS = -c -Wall -I$(INC_DIR)
exe:=main
DEPS = ${wildcard *.h}
SRC=${wildcard *.c}
#OBJ=main.o funA.o funB.o
OBJ=${SRC:.c=.o}
#$@ 工作目標名稱
#$< 第一個必條件的檔名
#$^ 所有必要條件的檔名
#$* 工作目標的主檔名

main: $(OBJ)
        ${CC} $(OBJ) $(HEAD) -I include  -o $@
        @echo "\nPhase2 -> Create executable file and clean includes\n"
        @sleep 0.5
        mv ${OBJ} ./../build/
        mv ${exe} ./../exe
#       rm ./*.h
.o:$(SRC) $(DEPS)
        @${CC} -o $@ $< $(CFLAGS)
```

main.c　　Makefile

funA.c

```c
/*funC.c*/
#include"funC.h"
extern float _main_base;
extern float _main_total;
static struct _discount{
        int Tag;
        float Total;
        float Seed[20];
        void(*set_Tag)();
        void(*set_Total)();
        pthread_mutex_t mutex;

} Discount = {
        19,
        0.0,
        {0.1,0.5,0.5,0.6,0.6,0.6,0.7,0.7,0.7,0.7,0.8,0.8,0.8,0.8,0.8,0.9,0.9,0.9,0.9,0.9},
        set_Tag,
        set_Total,
        PTHREAD_MUTEX_INITIALIZER
};
void set_Tag(){
        pthread_mutex_lock(&Discount.mutex);
        srand(time(NULL));
```

funB.c

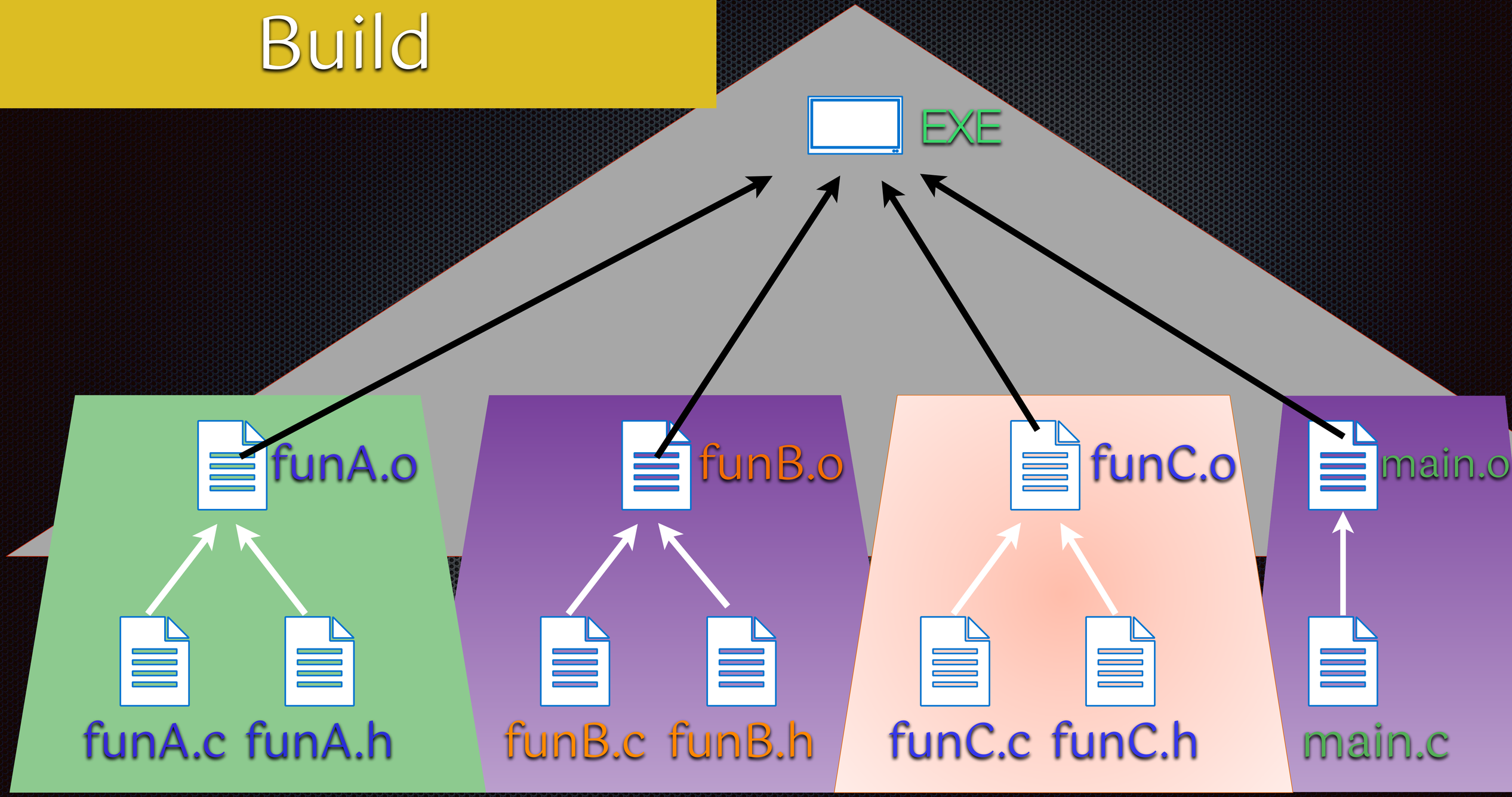
```c
/*funB.c*/
#include<stdio.h>
#include"funB.h"
extern float _main_base;
float _main_total;
static struct _item{
        float _shopping_cart;
        pthread_mutex_t mutex;
        void(*set_item_price)(float);
        float(*get_item_price)();
} Item = {
        45.99,
        PTHREAD_MUTEX_INITIALIZER,
        set_item_price,
        get_item_price
};
void set_item_price(float price){
        pthread_mutex_lock(&Item.mutex);
        Item._shopping_cart = price;
        pthread_mutex_unlock(&Item.mutex);
}

float get_item_price(){
        return Item._shopping_cart;
```

funC.c

```c
/*funA.c*/
#include"funA.h"
extern float _main_base;
static struct _account{
        float base;
        pthread_mutex_t mutex;
        void(*set_account_base)(float);
        float(*get_account_base)();
} Account = {
        0.0,
        PTHREAD_MUTEX_INITIALIZER,
        set_account_base,
        get_account_base
};
void set_account_base(float value){
        pthread_mutex_lock(&Account.mutex);
        Account.base = value;
        pthread_mutex_unlock(&Account.mutex);
}
float get_account_base(){
        return Account.base;
}
void funA(){
        float account_money =99.99;//init
```

Build


EXE
funA.o
funB.o
funC.o
main.o
funA.c
funA.h
funB.c
funB.h
funC.c
funC.h
main.c

More Illustration

I. ChatGPT

II. Sample Code

III. Document

IV. Youtube Tutorial

Advantages

I. Faster

II. Powerful

III. Easy to maintain

IV. Clear in file structure

V. Become more productive

Reference

I. GNU_Make

II. Wiki

III. Oracle

IV. Csdn

V. Mit.edu

VI. Www

VII. Gnu.org

VIII. Gitbook

IX. 知乎

X. Hackmd

XI. Mropengate

XII. Geeksforgeeks

XIII. Opengenus

XIV. YT

XV. YT2

XVI. Github

XVII. Cdsn2

XVIII. Cdsn3

XIX. Google_make

XX. Passlab.github

XXI. Wiki_ELF

XXII. Ivanagyro_ELF

XXIII. Blogspot

Questions?