# Introduction to 8086 Assembly

Lecture 11

Modular Programming in C and Assembly

### Modular Programming



#### test.c

```
#include <stdio.h>
extern int fact(int);
extern int maxval;
int main() {
 int x = 8;
 printf("mv=%d\n", maxval);
 printf("x!=%d\n", fact(x));
  return 0;
```

#### fact.c

```
int maxval = 2;
static int flag = 1;

int fact(int n) {
   return n==0 ? 1 : n*fact(n-1);
}

static int condmax(int a, int b) {
   return (a > b && flag) ? a : b;
}
```

### Modular Programming



#### test.c

```
#include <stdio.h>
int fact(int);
extern int maxval;
int main() {
 int x = 8;
 printf("mv=%d\n", maxval);
 printf("x!=%d\n", fact(x));
  return 0;
```

#### fact.c

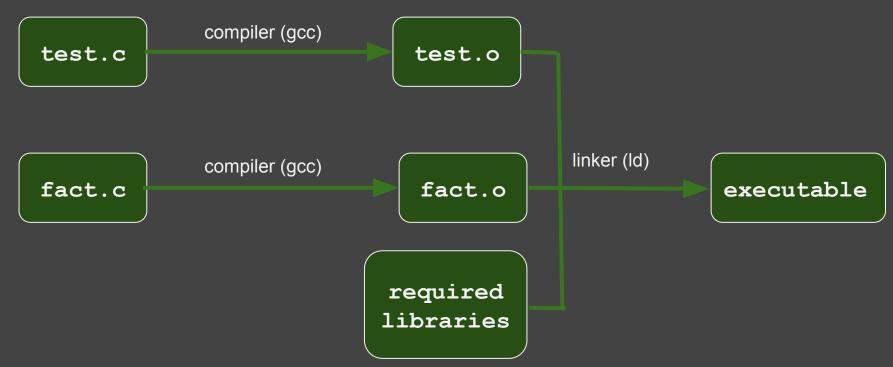
```
int maxval = 2;
static int flag = 1;

int fact(int n) {
   return n==0 ? 1 : n*fact(n-1);
}

static int condmax(int a, int b) {
   return (a > b && flag) ? a : b;
}
```

### Remember: Compiling and linking C files





### Remember: Compiling and linking C files



K. N. Toosi

```
test.c gcc -c test.c compile to
```

test.o

fact.c

gcc -c fact.c

compile to object file

object file

fact.o

required libraries

ld -dynamic-linker
/lib64/ld-linux-x86-64.so.2
/usr/lib/x86\_64-linux-gnu/crt1.o
/usr/lib/x86\_64-linux-gnu/crti.o
test.o fact.o
/usr/lib/x86\_64-linux-gnu/libc.so
/usr/lib/x86\_64-linux-gnu/crtn.o

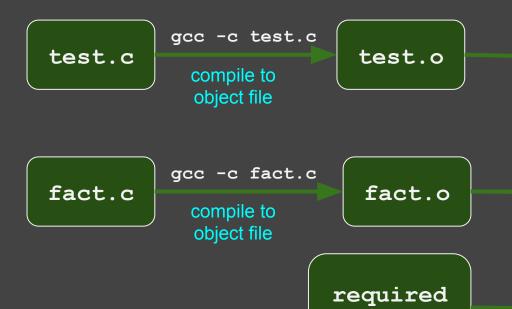
linking

executable

### Remember: Compiling and linking C files

libraries





easier: let gcc do the linking
gcc test.o fact.o
executable

gcc runs ld with appropriate arguments

### 32 bit Compiling and linking C files



```
test.c
```

```
compile to object file
```

test.o

fact.c

```
gcc -m32 -c fact.c

compile to
object file
```

fact.o

required libraries

```
ld -m elf_i386 -dynamic-linker
/lib/ld-linux.so.2
/usr/lib32/crt1.o
/usr/lib32/crti.o test.o fact.o
/usr/lib32/libc.so
/usr/lib32/crtn.o
```

executable

liking (32 bit)

### 32 bit Compiling and linking C files

required

libraries



test.c

compile to object file

gcc -m32 -c test.c

test.o

test.o

fact.c

fact.c

compile to object file

fact.o

easier: let gcc do the linking

gcc -m32 test.o fact.o

test

gcc runs ld with appropriate arguments

### 32 bit Compiling and linking C files



test.c compile to object file test.c

fact.c gcc -m32 -c fact.c fact.o compile to object file

required libraries

easier: let gcc do the linking

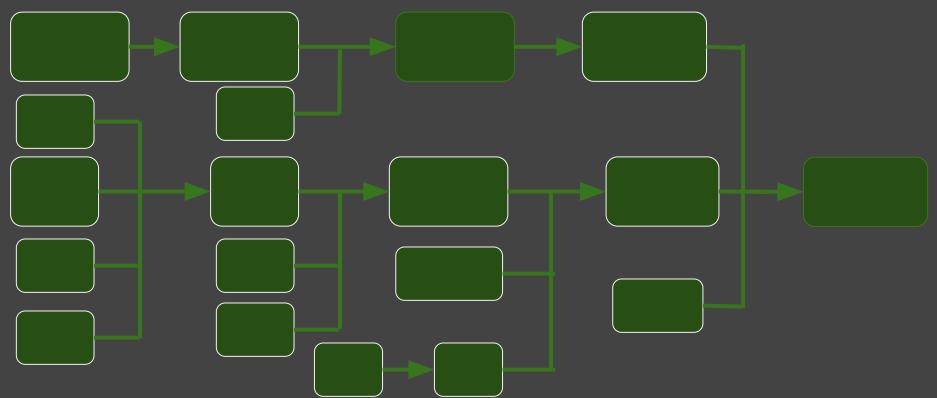
gcc -m32 test.o fact.o

test

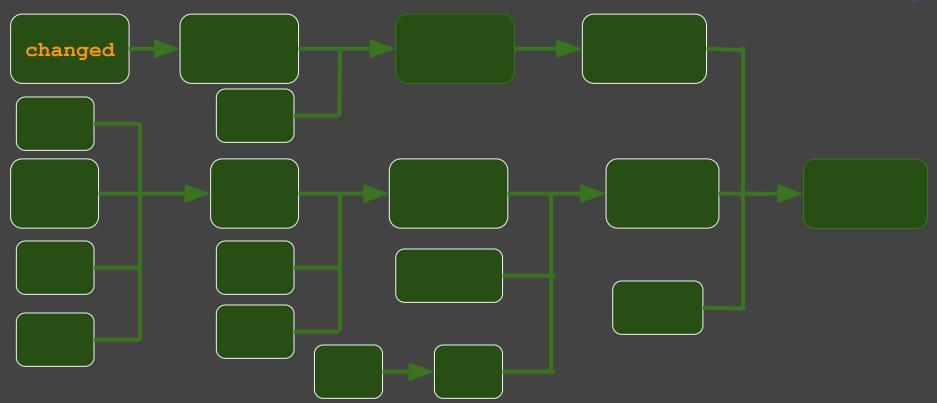
gcc runs ld with appropriate arguments

why not gcc -m32 test.c fact.c?

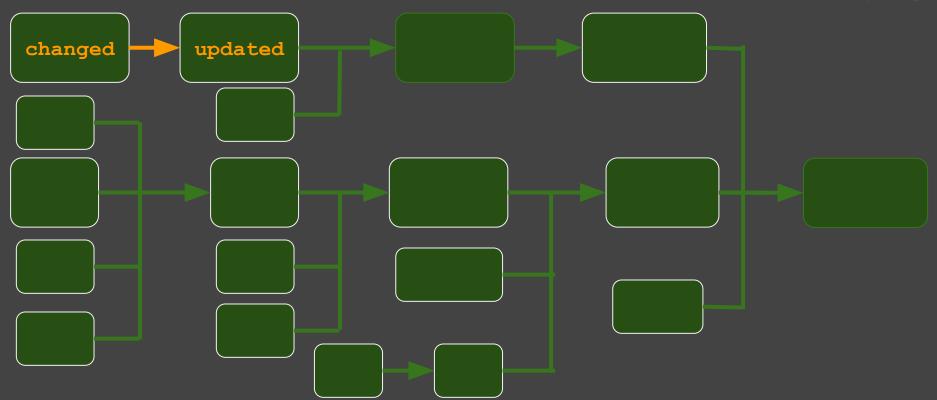




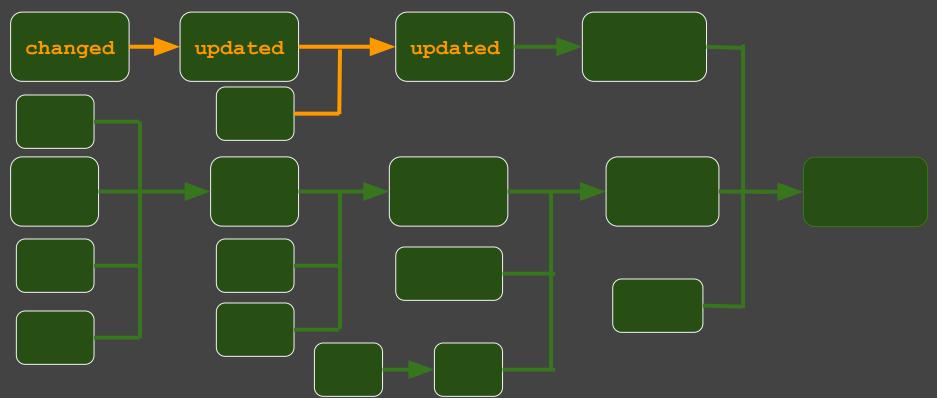




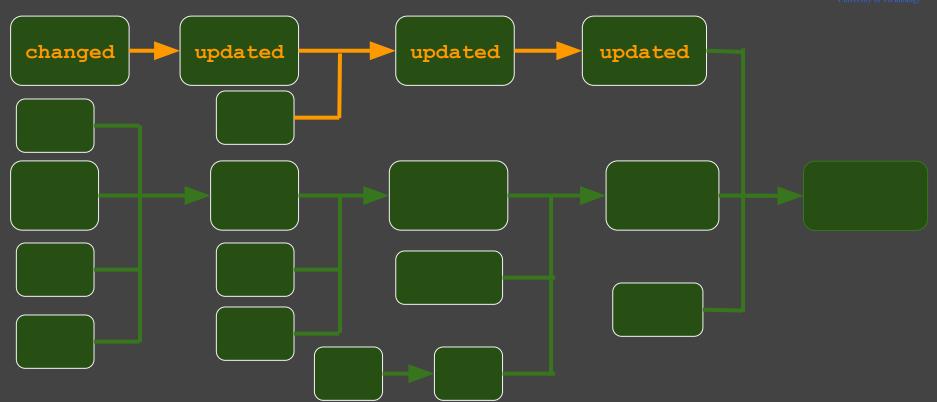




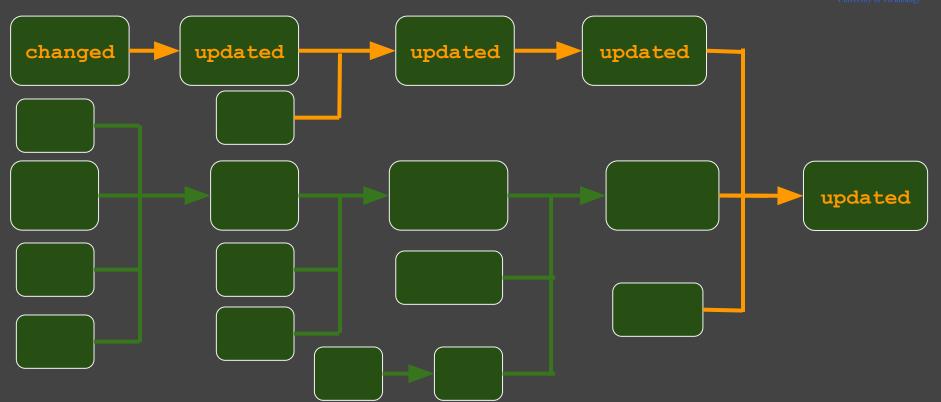




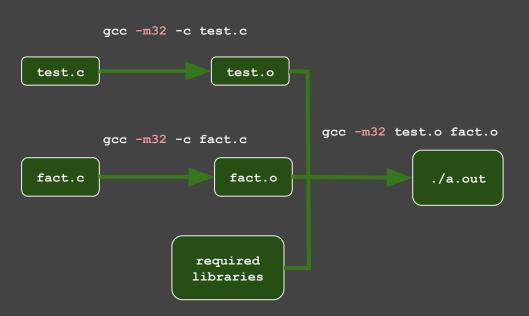


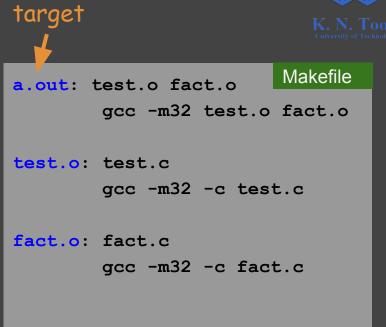








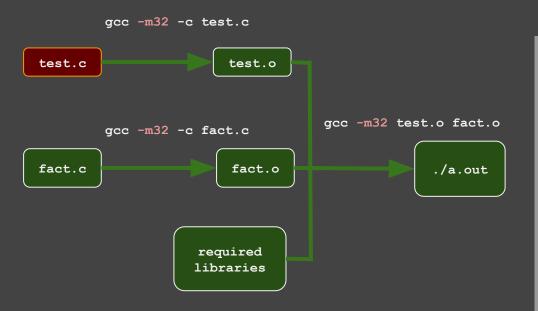


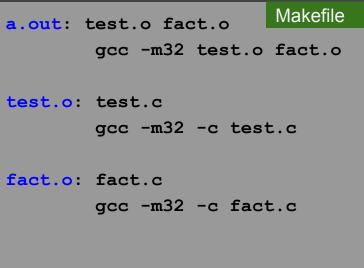




```
gcc -m32 -c test.c
                                                                              Makefile
                                                   a.out: test.o fact.o
test.c
                   test.o
                                                            gcc -m32 test.o fact.o
                            gcc -m32 test.o fact.o
      gcc -m32 -c fact.c
                                                   test.o: test.c
                                                            gcc -m32 -c test.c
fact.c
                   fact.o
                                        ./a.out
                                                   fact.o: fact.c
                                                            gcc -m32 -c fact.c
                required
               libraries
                                 b.nasihatkon@kntu:modular_c$ make
                                 gcc -m32 -c test.c
                                 gcc -m32 -c fact.c
                                 gcc -m32 test.o fact.o
```

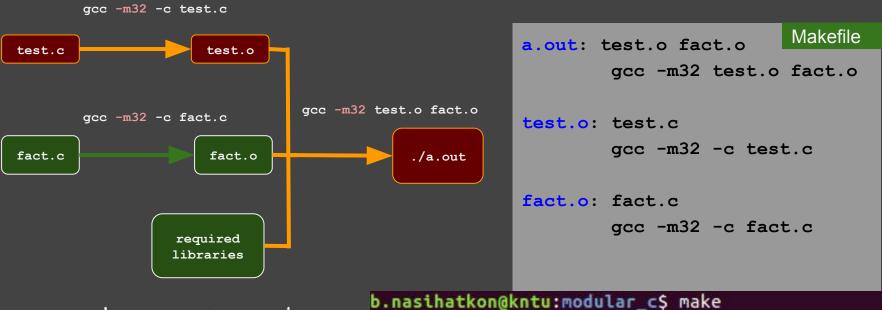






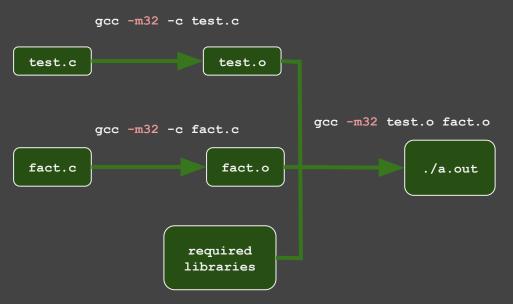
change test.c and run make again:





change **test.c** and run gcc -m32 -c test.c make again: gcc -m32 test.o fact.o

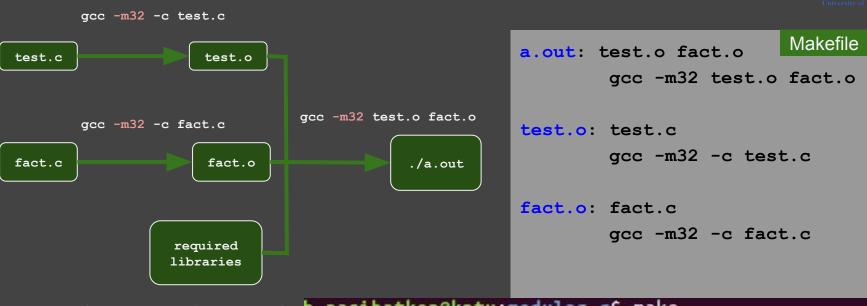




change nothing and rerun make:

```
Makefile
a.out: test.o fact.o
        gcc -m32 test.o fact.o
test.o: test.c
        gcc -m32 -c test.c
fact.o: fact.c
        gcc -m32 -c fact.c
```





change nothing and rerun make:

b.nasihatkon@kntu:modular\_c\$ make
make: 'a.out' is up to date.



- More on Makefile
  - http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
  - https://www.tutorialspoint.com/makefile/

### Modular Programming in assembly



- Multiple object files
- We have already done it!

### We have already done it!



driver.c

compile to object file

driver.o

myprog.asm

assemble to object file

myprog.o

asm\_io.asm

assemble to object file

asm\_io.o

C libraries

Link

executable

### We have already done it!



```
driver.c
```

```
gcc -m32 -c driver.c
```

driver.o

compile to object file

nasm -f elf myprog.asm

myprog.asm

assemble to object file

myprog.o

nasm -f elf -d ELF\_TYPE asm\_io.asm

asm\_io.asm

assemble to object file

asm\_io.o

C libraries

gcc -m32 driver.o myprog.o asm\_io.o

Link

executable

### Modular Programming in Assembly



```
first.asm
```

```
extern fact, var1
segment .text
    mov eax, [var1]
    ;; compute fact(6)
    push 6
    call fact
    add esp, 4
```

#### second.asm

```
global fact, var1
segment .data
var1: dd 22
segment .text
fact:
    :: factorial function
```

### Modular Programming in Assembly



```
first.asm
extern fact, var1
segment .text
    mov eax, [var1]
    ;; compute fact(6)
    push 6
    call fact
    add esp, 4
```

```
global fact, var1
segment .data
var1: dd 22
segment .text
fact:
    :: factorial function
```

second.asm

```
first.asm
%include "asm_io.inc"
segment .text
global asm_main
extern fact, var1
asm_main:
     pusha
     mov eax, [var1]
     call print_int
     call print_nl
    ;; compute fact(6)
    push 6
     call fact
     add esp, 4
     call print_int
     call print_nl
     popa
     ret
```

```
second.asm
global fact, var1
segment .data
var1: dd 22
segment .text
fact
    enter 0.0
     mov eax, [ebp+8]
     cmp eax, 0
    jg recur
     mov eax, 1
    jmp endfact
recur
     dec eax
     push eax
     call fact
     add esp, 4
    imul dword [ebp+8]
endfact:
     leave
     ret
```



```
first.asm
%include "asm_io.inc"
segment .text
global asm_main
extern fact, var1
asm_main:
     pusha
     mov eax, [var1]
    call print_int
     call print_nl
     ;; compute fact(6)
     push 6
     call fact
     add esp, 4
     call print_int
     call print_nl
     popa
     ret
```

```
second.asm
global fact, var1
segment .data
var1: dd 22
segment .text
fact
    enter 0.0
    mov eax, [ebp+8]
    cmp eax, 0
    jg recur
    mov eax, 1
    jmp endfact
recur
    dec eax
    push eax
    call fact
    add esp, 4
    imul dword [ebp+8]
endfact:
    leave
    ret
```



```
first.asm
%include "asm_io.inc"
segment .text
global asm_main
extern fact, var1
asm_main:
     pusha
     mov eax, [var1]
     call print_int
     call print_nl
    ;; compute fact(6)
     push 6
     call fact
     add esp, 4
     call print_int
     call print_nl
     popa
     ret
```

```
second.asm
     al fact, var1
segment .data
var1: dd 22
segment .text
fact
    enter 0.0
     mov eax, [ebp+8]
     cmp eax, 0
    jg recur
     mov eax, 1
    jmp endfact
recur
     dec eax
     push eax
     call fact
     add esp, 4
    imul dword [ebp+8]
endfact:
     leave
     ret
```



global fact, var1 segment .data

var1: dd 22

segment .text

fact:



second.asm

%include "asm\_io.inc" segment .text

How to assemble & link?

- \$ nasm -f elf first.asm
- \$ nasm -f elf second.asm

first.asm

\$ gcc -m32 -o first driver.c first.o second.o asm io.o

How to run?

./first

call print\_nl popa ret

endfact: leave

ret

### Using Makefile



```
Makefile
GCC OPTIONS= -m32
NASM OPTIONS= -f elf
first: driver.o first.o second.o asm io.o
      gcc $(GCC OPTIONS) -o first driver.o first.o second.o asm io.o
first.o: first.asm asm io.inc
      nasm $(NASM OPTIONS) first.asm
second.o: second.asm asm io.inc
      nasm $(NASM OPTIONS) second.asm
asm io.o: asm io.asm
      nasm $(NASM OPTIONS) -d ELF TYPE asm io.asm
driver.o: driver.c
      gcc $(GCC OPTIONS) -c driver.c
```

### Using Makefile



```
Makefile
GCC OPTIONS= -m32
NASM OPTIONS= -f elf
first: driver.o first.o second.o asm io.o
      qcc $(GCC OPTIONS) -o first driver.o first.o second.o asm io.o
                          b.nasihatkon@kntu:lecture11$ ls
first.o: first.asm asm ic1.html
                                    driver.c
                                                    Makefile
                                                                 second.asm
                          asm io.asm first modular_c second.asm.html
      nasm $ (NASM_OPTIONSasm io.asm.html first.asm README
                                                                  template.asm
                                     first.asm.html run.sh
                                                                  template.c
                          asm io.inc
                          b.nasihatkon@kntu:lecture11$ make
second.o: second.asm asm
                         gcc -m32 -c driver.c
      nasm $ (NASM_OPTIONS nasm -f elf first.asm
                          nasm -f elf second.asm
                          nasm -f elf -d ELF TYPE asm io.asm
asm io.o: asm io.asm
                          gcc -m32 -o first driver.o first.o second.o asm io.o
      nasm $ (NASM OPTIONS
                          b.nasihatkon@kntu:lecture11$ ./first
driver.o: driver.c
                          720
                          b.nasihatkon@kntu:lecture11$ make
      gcc $ (GCC OPTIONS)
                          make: 'first' is up to date.
                          b.nasihatkon@kntu:lecture11$
```

```
first.asm
%include "asm_io.inc"
segment .text
global asm_main
extern fact, var1
asm_main:
     pusha
     mov eax, [var1]
     call print_int
     call print nl
     ;; compute fact(6)
     push 6
     call fact
     add esp, 4
    call print_int
     call print_nl
     popa
     ret
```

```
global fact, var1
segment .data
var1: dd 22
segment .text
fact:
```



Why have not print\_int and
print nl been defined as extern?

second.asm

endfact: leave

ret

```
first.asm
%include "asm_io.inc"
segment .text
global asm_main
extern fact, var1
asm main:
    pusha
    mov eax, [var1]
    call print_int
    call print nl
    ;; compute fact(6)
    push 6
    call fact
    add esp, 4
    call print int
    call print_nl
    popa
    ret
```

```
global fact, var1
segment .data
var1: dd 22
segment .text
fact:
```

second.asm



Why have not print\_int and
print nl been defined as extern?

• Look at asm\_io.inc

```
endfact:
leave
ret
```



```
first.asm
%include "asm_io.inc"
seament .text
global asm_main
extern fact, var1
asm main:
    pusha
    mov eax, [var1]
    call print_int
    call print nl
    ;; compute fact(6)
    push 6
    call fact
    add esp, 4
    call print int
    call print_nl
    popa
    ret
```

```
asm io.inc
extern read_int, print_int, print_uint, print_string
extern read_char, print_char, print_nl
extern sub_dump_regs, sub_dump_mem, sub_dump_math, sub_dump_stack
%macro dump_regs 1
           dword %1
    push
    call
          sub_dump_regs
%endmacro
; usage: dump_mem label, start-address, # paragraphs
%macro dump_mem 3
    push dword %1
         dword %2
    push
    push dword %3
         sub dump mem
%endmacro
```

## Standalone assembly programs (32-bit)



prog1.asm

assemble to object file

prog1.o

prog2.asm

assemble to object file

prog2.o

Link

executable

prog3.asm

assemble to object file

prog3.o

### Standalone assembly programs (32-bit)

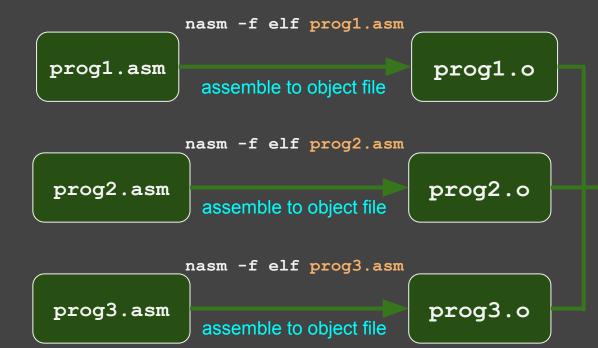


```
nasm -f elf prog1.asm
prog1.asm
                                      prog1.o
               assemble to object file
             nasm -f elf prog2.asm
                                     prog2.o
prog2.asm
               assemble to object file
             nasm -f elf prog3.asm
prog3.asm
                                     prog3.o
               assemble to object file
```

Link

### Standalone assembly programs (32-bit)





ld -m elf\_i386 prog1.o
prog2.o prog3.o

executable
Link

### Standalone assembly programs (64-bit)



```
nasm -f elf64 prog1.asm
                                      prog1.o
prog1.asm
               assemble to object file
             nasm -f elf64 prog2.asm
prog2.asm
                                     prog2.o
               assemble to object file
             nasm -f elf64 prog3.asm
                                     prog3.o
prog3.asm
               assemble to object file
```

```
ld prog1.o prog2.o prog3.o

Executable
```