

CS 33

Linkers

gcc Steps

1) Compile

- to start here, supply .c file
- to stop here: `gcc -S` (produces .s file)
- if not stopping here, gcc compiles directly into a .o file, bypassing the assembler

2) Assemble

- to start here, supply .s file
- to stop here: `gcc -c` (produces .o file)

3) Link

- to start here, supply .o file

The Linker

- **An executable program is one that is ready to be loaded into memory**
- **The linker (known as ld: /usr/bin/ld) creates such executables from:**
 - **object files produced by the compiler/assembler**
 - **collections of object files (known as libraries or archives)**
 - **and more we'll get to soon ...**

Linker's Job

- **Piece together components of program**
 - arrange within address space
 - » code (and read-only data) goes into text region
 - » initialized data goes into data region
 - » uninitialized data goes into bss region
- **Modify address references, as necessary**

A Program

```
int nprimes = 100;
```

data

```
int *prime, *prime2;
```

bss

```
int main() {
```

```
    int i, j, current = 1;
```

```
    prime = (int *)malloc(nprimes*sizeof(*prime));
```

```
    prime2 = (int *)malloc(nprimes*sizeof(*prime2));
```

dynamic

```
    prime[0] = 2; prime2[0] = 2*2;
```

```
    for (i=1; i<nprimes; i++) {
```

```
        NewCandidate:
```

```
            current += 2;
```

```
            for (j=0; prime2[j] <= current; j++) {
```

```
                if (current % prime[j] == 0)
```

```
                    goto NewCandidate;
```

```
            }
```

```
            prime[i] = current; prime2[i] = current*current;
```

```
        }
```

```
    return 0;
```

```
}
```

text

... with Output

```
int nprimes = 100;
int *prime, *prime2;
int main() {
    ...
    printcol(5);
    return 0;
}
```

```
void printcol(int ncols) {
    int i, j;
    int nrows = (nprimes+ncols-1)/ncols;
    for (i = 0; i<nrows; i++) {
        for (j=0; (j<ncols) && (i+nrows*j < nvals); j++) {
            printf("%6d", prime[i + nrows*j]);
        }
        printf("\n");
    }
}
```

... Compiled Separately

should refer to same thing

```
int nprimes = 100;
int *prime, *prime2;
int main() {
    ...
    printcol(5);
    return 0;
}
```

primes.c

ditto

```
extern int nprimes;
int *prime;
void printcol(int ncols) {
    int i, j;
    int nrows = (nprimes+ncols-1)/ncols;
    for (i = 0; i<nrows; i++) {
        for (j=0; (j<ncols)
            && (i+nrows*j < nvals); j++) {
            printf("%6d", prime[i + nrows*j]);
        }
        printf("\n");
    }
}
```

printcol.c

gcc -c primes.c

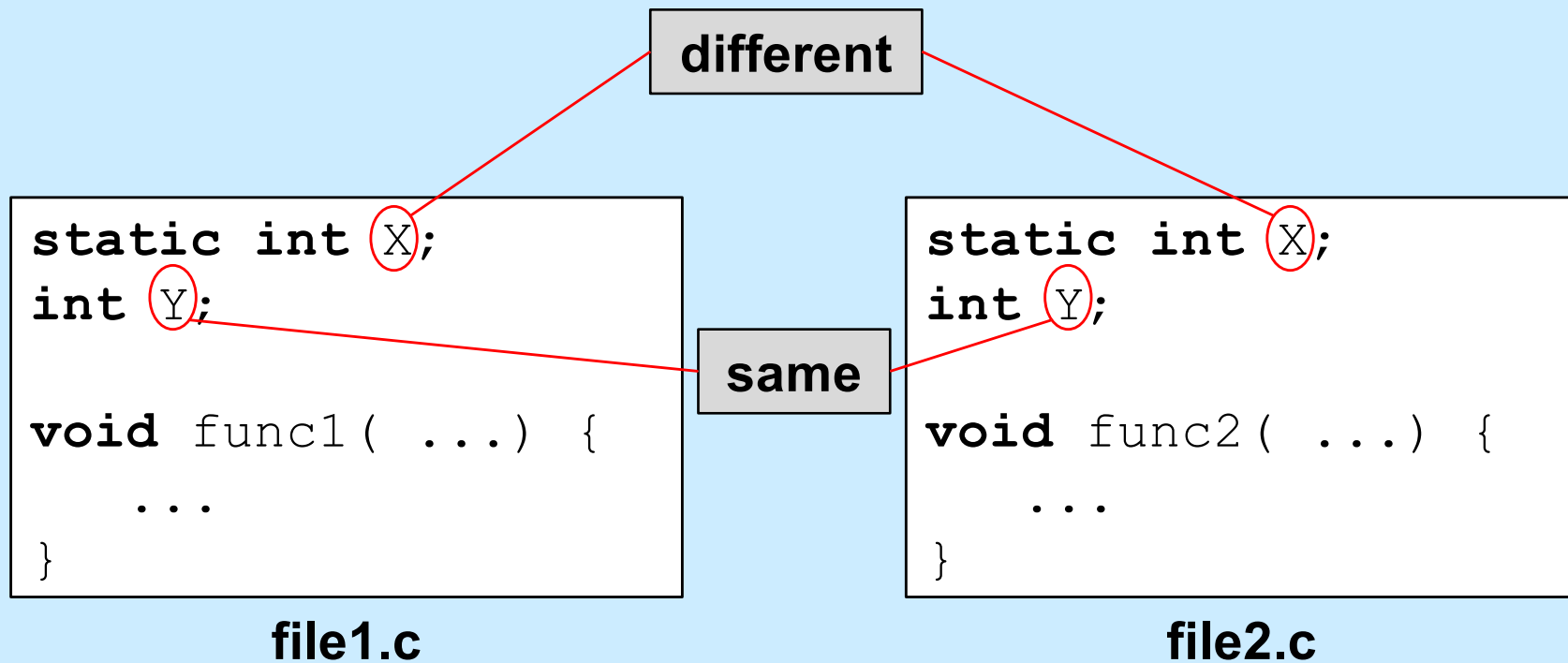
gcc -c printcol.c

gcc -o primes primes.o printcol.o

Global Variables

- **Initialized vs. uninitialized**
 - initialized allocated in *data* section
 - uninitialized allocated in *bss* section
 - » implicitly initialized to zero
- **File scope vs. program scope**
 - *static* global variables known only within file that declares them
 - » two of same name in different files are different
 - » e.g., `static int X;`
 - non-static global variables potentially shared across all files
 - » two of same name in different files are same
 - » e.g., `int X;`

Scope



Static Local Variables

```
int *sub1() {  
    int var = 1;  
    ...  
    return &var;  
    /* amazingly illegal */  
}
```

```
int *sub2() {  
    static int var = 1;  
    ...  
    return &var;  
    /* (amazingly) legal */  
}
```

Reconciling Program Scope (1)

tentative definition

```
int X;  
  
void func1( ...) {  
    ...  
}
```

file1.c

(complete) definition

```
int X=1;  
  
void func2( ...) {  
    ...  
}
```

file2.c

Where does X go?
What's its initial value?

- tentative definitions overridden by compatible (complete) definitions
- if not overridden, then initial value is zero

Reconciling Program Scope (2)

```
int X=2;  
  
void func1( ...) {  
    ...  
}
```

file1.c

```
int X=1;  
  
void func2( ...) {  
    ...  
}
```

file2.c

What happens here?

Reconciling Program Scope (3)

```
int X=1;  
  
void func1( ...) {  
    ...  
}
```

file1.c

```
int X=1;  
  
void func2( ...) {  
    ...  
}
```

file2.c

Is this ok?

Reconciling Program Scope (4)

```
extern int X;  
  
void func1( ...) {  
    ...  
}
```

file1.c

```
int X=1;  
  
void func2( ...) {  
    ...  
}
```

file2.c

What's the purpose of “extern”?

Default Values (1)

```
float seed = 1.0;
```

```
int PrimaryFunc(float arg) {  
    ...  
    SecondaryFunc(arg + seed);  
    ...  
}
```

```
void SecondaryFunc(float arg) {  
    ...  
}
```

Default Values (2)

```
float seed = 2.0; /* want a different seed */
```

```
int main() {  
    ...  
    PrimaryFunc(floatingValue);  
    ...  
}
```

```
void SecondaryFunc(float arg) {  
    /* would like to override default version */  
    ...  
}
```


Default Values (3)

```
__attribute__((weak)) float seed = 1.0;
```

```
int PrimaryFunc(float arg) {  
    ...  
    SecondaryFunc(arg + seed);  
    ...  
}
```

```
void __attribute__((weak)) SecondaryFunc(float arg) {  
    ...  
}
```

Does Location Matter?

```
int main(int argc, char *[]) {  
    return(argc);  
}
```

main:

```
pushq %rbp    ; push frame pointer  
movq  %rsp, %rbp    ; set frame pointer to point to new frame  
movl  %edi, %eax    ; put argc into return register (eax)  
movq  %rbp, %rsp    ; restore stack pointer  
popq  %rbp    ; pop stack into frame pointer  
ret         ; return: pops end of stack into rip
```

Location Matters ...

```
int X=6;  
int *aX = &X;
```

```
int main() {  
    void subr(int);  
    int y=*aX;  
    subr(y);  
    return(0);  
}
```

```
void subr(int i) {  
    printf("i = %d\n", i);  
}
```

Coping

- **Relocation**

- modify internal references according to where module is loaded in memory
- modules needing relocation are said to be *relocatable*
 - » which means they *require* relocation
- the compiler/assembler provides instructions to the linker on how to do this

A Revised Version of Our Program

```
extern int X;  
int *aX = &X;  
int Y = 1;  
  
int main() {  
    void subr(int);  
    int y = *aX+Y;  
    subr(y);  
    return(0);  
}
```

main.c

```
#include <stdio.h>  
int X;  
  
void subr(int XX) {  
    printf("XX = %d\n", XX);  
    printf("X = %d\n", X);  
}
```

subr.c

```
gcc -o prog -O1 main.c subr.c
```

main.s (1)

```
        .file      "main.c"
0:      .text
0:      .globl     main
0:      .type      main, @function
0: main:
0: .LFB0:
0:      .cfi_startproc
0:      subq       $8, %rsp
4:      .cfi_def_cfa_offset 16
4:      movq       aX(%rip), %rax
11:     movl       (%rax), %edi
13:     addl       Y(%rip), %edi
19:     call       subr
24:     movl       $0, %eax
29:     addq       $8, %rsp
33:     .cfi_def_cfa_offset 8
33:     ret
34:     .cfi_endproc
34: .LFE0:
34:     .size      main, .-main
```

**must be replaced with aX's
address, expressed as an offset
from the next instruction**

**must be replaced with Y's
address, expressed as an offset
from the next instruction**

**must be replaced with subr's
address, expressed as an offset
from the next instruction**

main.s (2)

```
0:      .globl  Y
0:      .data
0:      .align  4
0:      .type   Y, @object
0:      .size   Y, 4
0: Y:
0:      .long   1
4:      .globl  aX
8:      .align  8
8:      .type   aX, @object
8:      .size   aX, 8
8: aX:
8:      .quad   X
8:      .ident   "GCC: (Debian 4.7.2-5) 4.7.2"
0:      .section .note.GNU-stack,"",@progbits
```

Y should be made known to others

aX should be made known to others

must be replaced with address of X

subr.s (1)

```
        .file      "subr.c"
0:      .section    .rodata.str1.1,"aMS",@progbits,1
0: .LC0:
0:      .string     "XX = %d\n"
9: .LC1:
9:      .string     "X = %d\n"
```


subr.s (2)

```
0:      .text
0:      .globl subr
0:      .type   subr, @function

0: subr:
0: .LFB11:
0:      .cfi_startproc
0:      subq    $8, %rsp
4:      .cfi_def_cfa_offset 16
4:      movl    %edi, %esi
6:      movl    $.LC0, %edi
11:     movl    $0, %eax
16:     call    printf
21:     movl    X(%rip), %esi
27:     movl    $.LC1, %edi
32:     movl    $0, %eax
37:     call    printf
42:     addq    $8, %rsp
46:     .cfi_def_cfa_offset 8
46:     ret
47:     .cfi_endproc
47: .LFE11:
47:     .size    subr, .-subr
```

subr should be made known to others

must be replaced with .LC0's address

must be replaced with .LC1's address

must be replaced with printf's address, expressed as an offset from the next instruction

subr.s (3)

```
0:      .comm    X, 4, 4
0:      .ident   "GCC: (Debian 4.7.2-5) 4.7.2"
0:      .section .note.GNU-stack,"",@progbits
```

**reserve 4 bytes of 4-byte aligned
storage for X**

Quiz 1

```
int X;  
int proc(int arg) {  
    static int Y;  
    int Z;  
  
    ...  
}
```

Which of *X*, *Y*, *Z*, and *arg* would the compiler know the addresses of at compile time?

- a) all
- b) just *X* and *Y*
- c) just *arg* and *Z*
- d) none

ELF

- **Executable and linking format**
 - **used on most Unix systems**
 - » pretty much all but OS X
 - **defines format for:**
 - » **.o (object) files**
 - » **.so (shared object) files**
 - » **executable files**

Doing Relocation

- **Linker is provided instructions for updating object files**
 - **lots of ways addresses can appear in machine code**
 - **three in common use on x86-64**
 - » **32-bit absolute addresses**
 - **used for text references**
 - » **64-bit absolute addresses**
 - **used for data references**
 - » **32-bit PC-relative addresses**
 - **offset from current value of rip**
 - **used for text and data references**

main.o (1)

ELF Header:

```
Magic:    7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class:                                ELF64
Data:                                2's complement, little endian
Version:                               1 (current)
OS/ABI:                                UNIX - System V
ABI Version:                           0
Type:                                  REL (Relocatable file)
Machine:                               Advanced Micro Devices X86-64
Version:                               0x1
Entry point address:                   0x0
Start of program headers:               0 (bytes into file)
Start of section headers:               296 (bytes into file)
Flags:                                 0x0
Size of this header:                    64 (bytes)
Size of program headers:                0 (bytes)
Number of program headers:               0
Size of section headers:                64 (bytes)
Number of section headers:               13
Section header string table index: 10
```

main.o (2)

32-bit, PC-relative address

Relocation section '.rela.text' at offset 0x5c0 contains 3 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000007	0009000000002	R_X86_64_PC32	0000000000000008	aX - 4
0000000000000f	000a000000002	R_X86_64_PC32	0000000000000000	Y - 4
00000000000014	000b000000002	R_X86_64_PC32	0000000000000000	subr - 4

Relocation section '.rela.data' at offset 0x608 contains 1 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000008	000c000000001	R_X86_64_64	0000000000000000	X + 0

64-bit, absolute address

0:	48 83 ec 08	sub	\$0x8,%rsp	
4:	48 8b 05 00 00 00 00	mov	0x0(%rip),%rax	# b <main+0xb>
b:	8b 38	mov	(%rax),%edi	
d:	03 3d 00 00 00 00	add	0x0(%rip),%edi	# 13 <main+0x13>
13:	e8 00 00 00 00	callq	18 <main+0x18>	
18:	b8 00 00 00 00	mov	\$0x0,%eax	
1d:	48 83 c4 08	add	\$0x8,%rsp	
21:	c3	retq		

main.o (3)

Relocation section '.rela.text' at offset 0x5c0 contains 3 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000007	0009000000002	R_X86_64_PC32	0000000000000008	aX - 4
0000000000000f	000a000000002	R_X86_64_PC32	0000000000000000	Y - 4
00000000000014	000b000000002	R_X86_64_PC32	0000000000000000	subr - 4

Relocation section '.rela.data' at offset 0x608 contains 1 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000008	000c000000001	R_X86_64_64	0000000000000000	X + 0

0:	48 83 ec 08	sub	\$0x8,%rsp	
4:	48 8b 05 00 00 00 00	mov	0x0(%rip),%rax	# b <main+0xb>
b:	8b 38	mov	(%rax),%edi	
d:	03 3d 00 00 00 00	add	0x0(%rip),%edi	# 13 <main+0x13>
13:	e8 00 00 00 00	callq	18 <main+0x18>	
18:	b8 00 00 00 00	mov	\$0x0,%eax	
1d:	48 83 c4 08	add	\$0x8,%rsp	
21:	c3	retq		

main.o (4)

Relocation section '.rela.text' at offset 0x5c0 contains 3 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000007	0009000000002	R_X86_64_PC32	0000000000000008	aX - 4
0000000000000f	000a000000002	R_X86_64_PC32	0000000000000000	Y - 4
00000000000014	000b000000002	R_X86_64_PC32	0000000000000000	subr - 4

Relocation section '.rela.data' at offset 0x608 contains 1 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000008	000c000000001	R_X86_64_64	0000000000000000	X + 0

0:	48 83 ec 08	sub	\$0x8,%rsp	
4:	48 8b 05 00 00 00 00	mov	0x0(%rip),%rax	# b <main+0xb>
b:	8b 38	mov	(%rax),%edi	
d:	03 2d 00 00 00 00	add	0x0(%rip),%edi	# 13 <main+0x13>
13:	e8 00 00 00 00	callq	18 <main+0x18>	
18:	b8 00 00 00 00	mov	\$0x0,%eax	
1d:	48 83 c4 08	add	\$0x8,%rsp	
21:	c3	retq		

main.o (5)

Relocation section '.rela.text' at offset 0x5c0 contains 3 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000007	0009000000002	R_X86_64_PC32	0000000000000008	aX - 4
0000000000000f	000a000000002	R_X86_64_PC32	0000000000000000	Y - 4
00000000000014	000b000000002	R_X86_64_PC32	0000000000000000	subr - 4

Relocation section '.rela.data' at offset 0x608 contains 1 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000008	000c000000001	R_X86_64_64	0000000000000000	X + 0

0:	48 83 ec 08	sub	\$0x8,%rsp	
4:	48 8b 05 00 00 00 00	mov	0x0(%rip),%rax	# b <main+0xb>
b:	8b 38	mov	(%rax),%edi	
d:	03 3d 00 00 00 00	add	0x0(%rip),%edi	# 13 <main+0x13>
13:	e8 00 00 00 00	callq	18 <main+0x18>	
18:	b8 00 00 00 00	mov	\$0x0,%eax	
1d:	48 83 c4 08	add	\$0x8,%rsp	
21:	c3	retq		

main.o (6)

Relocation section '.rela.text' at offset 0x5c0 contains 3 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000007	0009000000002	R_X86_64_PC32	0000000000000008	aX - 4
0000000000000f	000a000000002	R_X86_64_PC32	0000000000000000	Y - 4
00000000000014	000b000000002	R_X86_64_PC32	0000000000000000	subr - 4

Relocation section '.rela.data' at offset 0x608 contains 1 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000008	000c000000001	R_X86_64_64	0000000000000000	X + 0

0:	48 83 ec 08	sub	\$0x8,%rsp	
4:	48 8b 05 00 00 00 00	mov	0x0(%rip),%rax	# b <main+0xb>
b:	8b 38	mov	(%rax),%edi	
d:	03 3d 00 00 00 00	add	0x0(%rip),%edi	# 13 <main+0x13>
13:	e8 00 00 00 00	callq	18 <main+0x18>	
18:	b8 00 00 00 00	mov	\$0x0,%eax	
1d:	48 83 c4 08	add	\$0x8,%rsp	
21:	c3	retq		

subr.o (1)

ELF Header:

```
Magic:    7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class:                                ELF64
Data:                                2's complement, little endian
Version:                               1 (current)
OS/ABI:                                UNIX - System V
ABI Version:                           0
Type:                                   REL (Relocatable file)
Machine:                               Advanced Micro Devices X86-64
Version:                               0x1
Entry point address:                   0x0
Start of program headers:               0 (bytes into file)
Start of section headers:               312 (bytes into file)
Flags:                                 0x0
Size of this header:                    64 (bytes)
Size of program headers:                 0 (bytes)
Number of program headers:                0
Size of section headers:                 64 (bytes)
Number of section headers:                13
Section header string table index: 10
```

subr.o (2)

Relocation section '.rela.text' at offset 0x5b0 contains 5 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
00000000000007	000500000000a	R_X86_64_32	0000000000000000	.rodata.str1.1 + 0
00000000000011	000a000000002	R_X86_64_PC32	0000000000000000	printf - 4
00000000000017	000b000000002	R_X86_64_PC32	0000000000000004	X - 4
0000000000001c	000500000000a	R_X86_64_32	0000000000000000	.rodata.str1.1 + 9
00000000000026	000a000000002	R_X86_64_PC32	0000000000000000	printf - 4

0:	48 83 ec 08	sub	\$0x8,%rsp	
4:	89 fe	mov	%edi,%esi	
6:	bf 00 00 00 00	mov	\$0x0,%edi	
b:	b8 00 00 00 00	mov	\$0x0,%eax	
10:	e8 00 00 00 00	callq	15 <subr+0x15>	
15:	8b 35 00 00 00 00	mov	0x0(%rip),%esi	# 1b <subr+0x1b>
1b:	bf 00 00 00 00	mov	\$0x0,%edi	
20:	b8 00 00 00 00	mov	\$0x0,%eax	
25:	e8 00 00 00 00	callq	2a <subr+0x2a>	
2a:	48 83 c4 08	add	\$0x8,%rsp	
2e:	c3	retq		

.rodata.str1.1:
XX = %d\n\0X = %d\n\0

Quiz 2

Consider the following 5-byte instruction:

ea 00 00 00 00

ea is the opcode for the call instruction with a 32-bit PC-relative operand.

**Suppose this instruction is at location 0x1000.
To what location would control be transferred if the instruction were executed as is?**

- a) 0**
- b) 0x1000**
- c) 0x1001**
- d) 0x1005**

printf.o

Relocation section '.rela.text' at offset 0x5c0 contains 3 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
0000000002d3	000b00000002	R_X86_64_PC32	0000000000000000	write - 4

Relocation section '.rela.data' at offset 0x608 contains 1 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
0000000000d3	000c00000001	R_X86_64_64	0000000000000000	StandardFiles + 0

prog

ELF Header:

```
Magic:    7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class:                                ELF64
Data:                                2's complement, little endian
Version:                               1 (current)
OS/ABI:                                UNIX - System V
ABI Version:                           0
Type:                                   EXEC (Executable file)
Machine:                               Advanced Micro Devices X86-64
Version:                               0x1
Entry point address:                   0x400400
Start of program headers:               64 (bytes into file)
Start of section headers:               2704 (bytes into file)
Flags:                                 0x0
Size of this header:                    64 (bytes)
Size of program headers:                 56 (bytes)
Number of program headers:                8
Size of section headers:                 64 (bytes)
Number of section headers:               31
Section header string table index: 28
```


Final Result

Symbol	Value	Size	
_start	0x400400	0x60	}
main	0x400460	0x3f	
subr	0x4004a0	0x30	
printf	0x4004d0	0x12000	
write	0x4124d0	0x30	
.rodata	0x412500	0x9	}
aX	0x413000	0x8	
Y	0x413008	0x8	
StandardFiles	0x413010	0x1000	
X	0x414010	0x8	}

text

data

bss