Introduction – Learning Goals

- Understanding the overall processes by which most major systems and infrastructure software is built and executed today
  - how symbols in high-level languages are resolved to addresses and constants in machine code
  - the coordination of linker and loader, particularly in systems exploiting virtual address spaces
  - how linkers allocate space for variables and functions, including the role of symbol tables
- Become proficient as a software engineer at the intermediate level in the C language when separate compilation is used for medium and large programs
  - separate compilation and the role of header files in creating modular code
  - common mistakes
  - best practices when declaring and defining functions and variables
  - the role and purpose of static libraries
  - the role and purpose of dynamic libraries
  - the purpose of whole program link-time optimization
  - the implications of virtual address space layout for debugging program faults
  - how to use common tools such as nm, objdump, etc.
Figure 1: Compilation, Linking, and Loading in a typical System
Preprocessor performs textual insertion of include files

Compiler resolves the following symbolic names:
- Local automatic variables, function parameters
- Field names in structures

Assembler resolves certain labels for relative branches

The resulting relocatable .o file still retains symbolic names for all functions and variables that are global in extent

```c
extern long longabs(long j);

struct Point {
    long x, y;
};

long manhattan(struct Point * p0, struct Point * p1)
{
    long dx = p0->x - p1->x;
    long dy = p0->y - p1->y;
    return longabs(dx) + longabs(dy);
}

// symbol table
U longabs
0000000000000000 T manhattan
```

```assembly
manhattan:
    pushq %rbp
    pushq %rbx
    subq $8, %rsp
    movq 8(%rdi), %rbp
    movq (%rdi), %rdi
    subq 8(%rsi), %rbp
    subq (%rsi), %rdi
    call longabs
    movq %rbp, %rdi
    movq %rax, %rbx
    call longabs
    addq $8, %rsp
    addq %rbx, %rax
    popq %rbx
    popq %rbp
    ret
```
Relocatable Object Files – Text Section

0000000000000000 <manhattan>:

0: 55 push %rbp
1: 53 push %rbx
2: 48 83 ec 08 sub $0x8,%rsp
6: 48 8b 6f 08 mov 0x8(%rdi),%rbp
a: 48 8b 3f mov (%rdi),%rdi
d: 48 2b 6e 08 sub 0x8(%rsi),%rbp
11: 48 2b 3e sub (%rsi),%rdi
14: e8 00 00 00 00 callq 19 <manhattan+0x19>
19: 48 89 ef mov %rbp,%rdi
1c: 48 89 c3 mov %rax,%rbx
1f: e8 00 00 00 00 callq 24 <manhattan+0x24>
24: 48 83 c4 08 add $0x8,%rsp
28: 48 01 d8 add %rbx,%rax
2b: 5b pop %rbx
2c: 5d pop %rbp
2d: c3 retq

OFFSET TYPE VALUE

0000000000000015 R_X86_64_PC32 longabs-0x0000000000000004
0000000000000020 R_X86_64_PC32 longabs-0x0000000000000004

- Contain multiple sections (only text shown here)
- Each section is laid out starting as if starting at 0
- Contains relocation records: placeholders and meta information about how to patch them up once actual addresses are known
ELF is a standard format for relocatable object files, executables, and shared objects (dynamic libraries) used in System V and derived systems, including Linux [URL].

Other formats include Mach-O (OSX), PE (Windows), a.out.

Provides the link between compiler → linker → loader.

Carries all information needed by the next tool; also debugging and exception handling information.

Extensive tool support.
extern long elong;

char *name = "CS3214";
long *ptr = &elong;
long s_long = 42;
long w_long;

long adder()
{
    return *ptr + s_long + w_long;
}

// symbol table
0000000000000000 T adder
0000000000000000 U elong
000000000000000010 D name
000000000000000008 D ptr
000000000000000000 D s_long
000000000000000008 C w_long

adder:
    movq ptr(%rip), %rdx
    movq s_long(%rip), %rax
    addq (%rdx), %rax
    addq w_long(%rip), %rax
    ret

.comm  w_long,8,8
.data
.s_long:
    .quad 42
.ptr:
    .quad 42

.section .rodata
.LC0:
    .string "CS3214"
.data
.name:
    .quad .LC0
Relocatable Object Files – Data, BSS, Read-only Section

0000000000000000 <adder>:
0:  48 8b 15 00 00 00 00     mov  0x0(%rip),%rdx
7:  48 8b 05 00 00 00 00     mov  0x0(%rip),%rax
  e:  48 03 02                 add  (%rdx),%rax
11: 48 03 05 00 00 00 00     add  0x0(%rip),%rax
18:  c3                          retq

OFFSET   TYPE              VALUE
0000000000000003  R_X86_64_PC32  ptr-0x0000000000000004
000000000000000a  R_X86_64_PC32  s_long-0x0000000000000004
0000000000000014  R_X86_64_PC32  w_long-0x0000000000000004

OFFSET   TYPE              VALUE
0000000000000008  R_X86_64_64  elong
0000000000000010  R_X86_64_64  .rodata.str1.1

Contents of section .rodata.str1.1:
0000 43533332 313400  CS3214.

- Global variables that have programmer-defined initial values are stored in the data section (readonly if constant)
- Global variables without programmer-defined initial values are listed in so-called BSS section ("better save space")
Multiple .o object files are merged into an executable by the linker.

This merging process creates an in-memory layout of the process’s code and data.

The linker resolves references (by matching them to definitions) and relocates symbols to their computed address and fills in any placeholders referring to them.

```c
long elong = 13;
long longabs(long j) {
    return j >= 0 ? j : -j;
}

int main()
{
    extern long adder();
    return adder();
}
```

```
// symbol table
U adder
0000000000000000 D elong
0000000000000000 T longabs
0000000000000000 T main
```
Resulting Symbol Table

source1.o

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>longabs</td>
<td>U</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>manhattan</td>
<td>T</td>
<td>0000000000000000</td>
</tr>
</tbody>
</table>

source2.o

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>adder</td>
<td>T</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>elong</td>
<td>U</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>name</td>
<td>D</td>
<td>0000000000000010</td>
</tr>
<tr>
<td>ptr</td>
<td>D</td>
<td>0000000000000008</td>
</tr>
<tr>
<td>s_long</td>
<td>D</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>w_long</td>
<td>C</td>
<td>0000000000000008</td>
</tr>
</tbody>
</table>

source3.o

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>adder</td>
<td>U</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>elong</td>
<td>D</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>longabs</td>
<td>T</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>main</td>
<td>T</td>
<td>0000000000000000</td>
</tr>
</tbody>
</table>

exe

// .text.startup
0000000000000000 T main

// .text
00000000000000400550 T manhattan
00000000000000400580 T adder
000000000000004005a0 T longabs

// .data
0000000000000601020 D s_long
0000000000000601028 D ptr
0000000000000601030 D name
0000000000000601038 D elong

// .bss
0000000000000601048 B w_long

Godmar Back

Linking & Loading I 10/15

VIRGINIA TECH.
The linker merges like-sections in sequential order (usually as provided on the command line)

Guided by linker script (`ld --verbose`)

Resulting executable is designed to be efficiently loaded (or mapped) at load time into the process’s virtual address space
Content of Final Executable

(showing excerpts from .text and .data sections)

```
0000000000400450 <main>:
  400450:  48 83 ec 08 sub $0x8,%rsp
  400454:  31 c0 xor %eax,%eax
  400456:  e8 25 01 00 00 callq 400580 <adder>
  40045b:  48 83 c4 08 add $0x8,%rsp
  40045f:  c3 retq

0000000000400550 <manhattan>:
  400550:  55 push %rbp
  400551:  53 push %rbx
  400552:  48 83 ec 08 sub $0x8,%rsp
  400556:  48 8b 6f 08 mov 0x8(%rdi),%rbp
  40055a:  48 8b 3f mov (%rdi),%rdi
  40055d:  48 2b 6e 08 sub 0x8(%rsi),%rbp
  400561:  48 2b 3e sub (%rsi),%rdi
  400564:  e8 37 00 00 00 callq 4005a0 <longabs>
  400569:  48 89 ef mov %rbp,%rdi
  40056c:  48 89 c3 mov %rax,%rbx
  40056f:  e8 2c 00 00 00 callq 4005a0 <longabs>
  400574:  48 83 c4 08 add $0x8,%rsp
  400578:  48 01 d8 add %rbx,%rax
  40057b:  5b pop %rbx
  40057c:  5d pop %rbp
  40057d:  c3 retq
  40057e:  66 90 xchg %ax,%ax

0000000000400580 <adder>:
  400580:  48 8b 15 a1 0a 20 00 mov 0x200a1(%rip),%rdx
  400587:  48 8b 05 92 0a 20 00 mov 0x200a2(%rip),%rax
  40058e:  48 03 02 add %rdx,%rax
  400591:  48 03 05 b0 0a 20 00 add 0x200ab0(%rip),%rax
  400598:  c3 retq

00000000004005a0 <longabs>:
  4005a0:  48 89 fa mov %rdi,%rdx
  4005a3:  48 89 f8 mov %rdi,%rax
  4005a6:  48 c1 fa 3f sar $0x3f,%rdx
  4005aa:  48 31 d0 xor %rdx,%rax
  4005b0:  c3 retq

0000000000601020 <s_long>:
  601020:  2a 00 00 00 00 00 00 00 00 00 00

0000000000601028 <ptr>:
  601028:  38 10 60 00 00 00 00 00 00

0000000000601030 <name>:
  601030:  58 0e 40 00 00 00 00 00 00 00 00

0000000000601038 <elong>:
  601038:  0d 00 00 00 00 00 00 00 00 00 00
```
Virtual Address Space Layout (Linux x86_64, 48-bit)

$ tac /proc/88988/maps

User Address Space (128TB)
Kernel Address Space
text
data
bss
heap
stack
various other objects:
shared libraries, mmapped files,
special purpose regions

User Address Space (128TB)
Address Space Layout Randomization (ASLR)

- To increase defenses against remote execution vulnerabilities modern systems try to randomize as much of their address space as possible
- E.g., stack, heap locations, but to an increasing extent also code + data
- This impacts the linking process: in general, linker-assigned addresses are “baked” into the machine code, which can be loaded directly
- Loaders can also perform load time relocation (at a cost)
- Position-Independent Code (PIC) can be loaded at any address in the address space without further relocation
- In general, PIC requires indirection. The x86-64’s IP-relative addressing mode disp(%rip) was introduced to assist in this.
Compiler resolves certain symbolic names, but passes any that are global in extent onto the linker as references in relocatable object files.

Linker merges object files to produce an executable, computing a virtual address space layout in the process.

The executable contains the text and data needed to load a program into memory.

We have ignored so far:

- Lexical scoping rules (global vs. local to a compilation unit)
- Rules the linker applies when deciding how to resolve an external reference
- Static and dynamic libraries