CSE278: Introduction to Systems Programming (Systems I)

Lab #9

Due: Mon/Tue May 4/5 during Lab time

Maximum Points: 50

Submission Instructions

This part of the homework assignment must be turned-in electronically via Canvas. Ensure you name this document `Lab9_MUID.docx`, where `MUid` is your Miami University unique ID. (Example: Lab9_ahmede.docx)

Copy pasting from online resources is **Plagiarism**. Instead you should read, understand, and use your own words to respond to questions.

Submission Instructions:
Once you have completed answering the questions save this document as a PDF file (**don't just rename the document; that is not the correct way to save as PDF**) and upload it to Canvas.

General Note: Upload each file associated with homework (or lab exercises) individually to Canvas. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

Objective

The objective of this Lab is to review basic concepts of:

- Linux ABI
- Debugging with gdb

Name: **Ben Hilger**

Required reading

- Lecture Slides & ClassNotes: Security
- Lecture Slides & ClassNotes: ComputerArchitecture

PART A: Linux Application Binary Interface (ABI)

Goals

- Details of how a program executes on a Unix/Linux system

.

Linux ABI defines most of the low-level details of a program including:

- The register layout (rip, rsp, rbp, rax,, rbx, rcx, rdx, rdi, rsi, r8, r9, r10, r11, r12, r13, r14, and r15)
- The stack frame
  - Pushing to the stack *subtracts* from the stack pointer
  - Popping from the stack *adds* to the stack pointer
  - call
  - ret
- Function prologs and epilogs
- The calling convention (that is, parameter passing)
- Exception handling
- Virtual memory layout
- The binary object format (ELF) (begins with 0x7F)

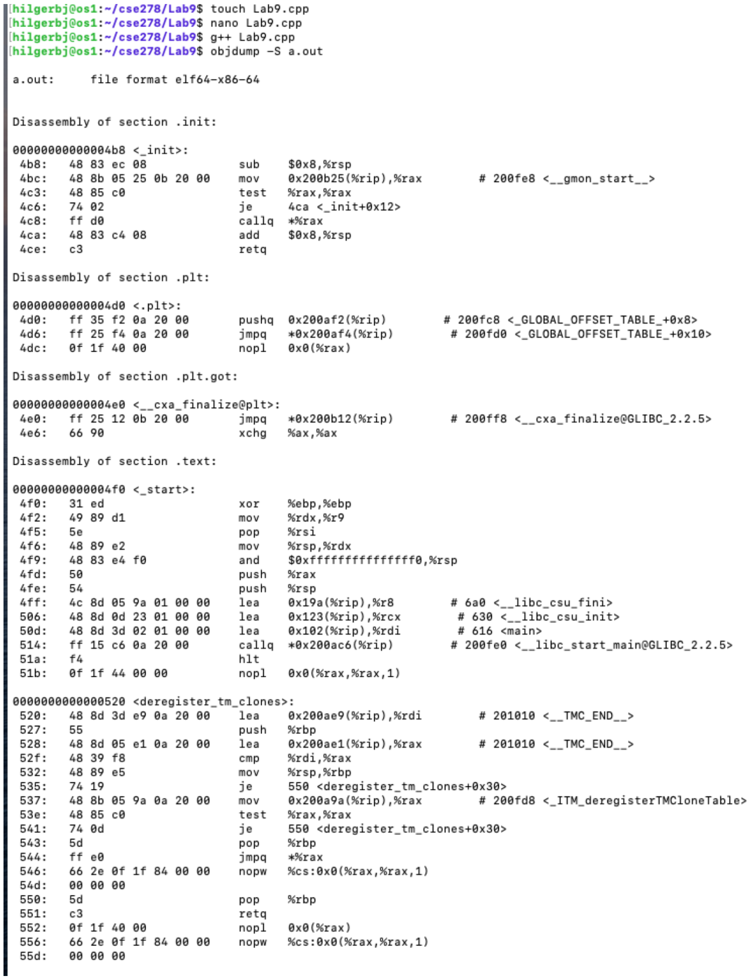
1. Consider the following code fragment:

```
int test()
{
      int i = 1;
      int j = 2;
      return i + j;
}
int main (void)
{
          test();
}
```

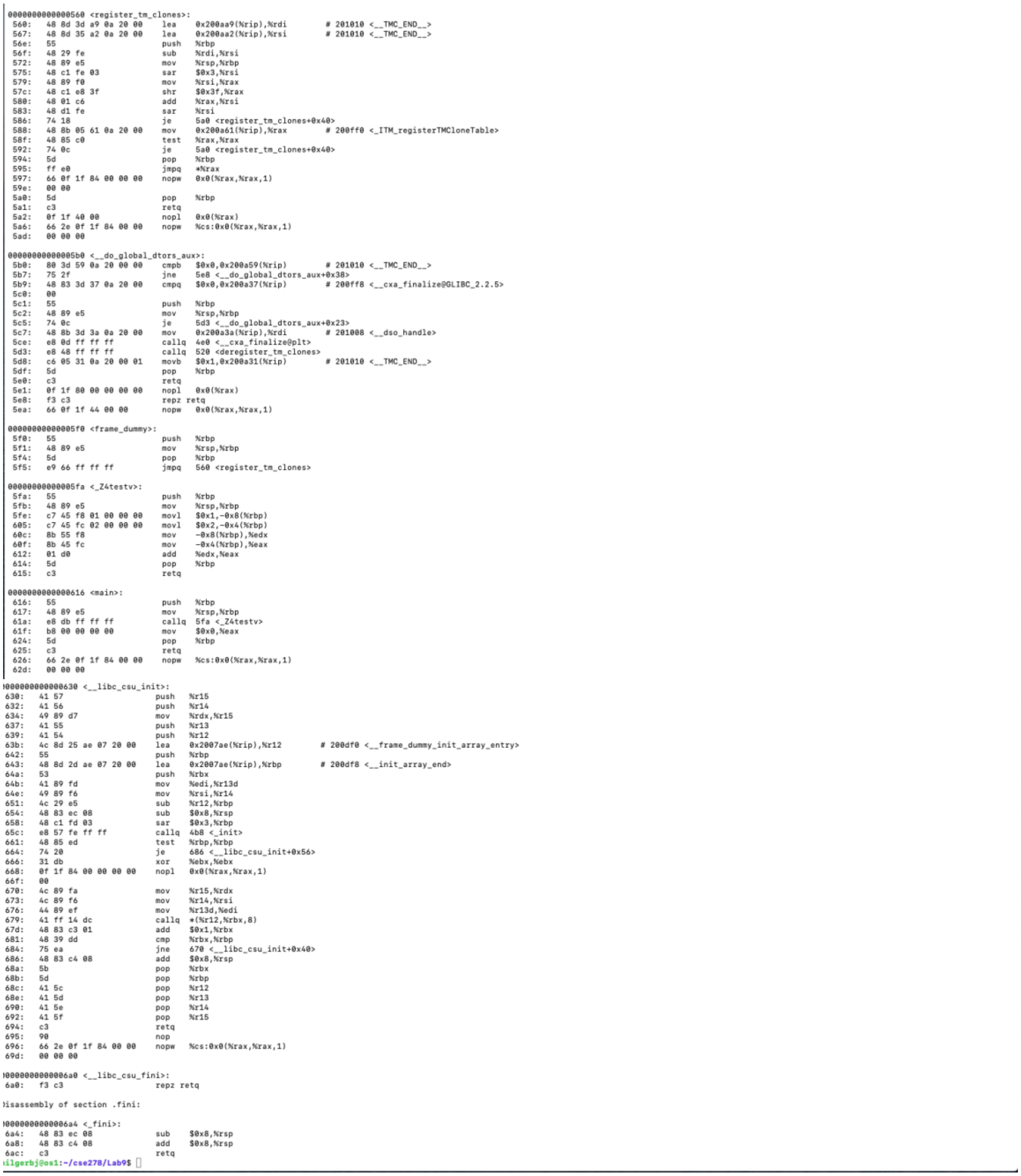
A. Compile the Lab9.cpp as follows:
g++ Lab9.cpp

B. Disassemble the resulting binary using the command
objdump –S a.out

C. Attach a screen shot



```
[hilgerbj@os1:~/cse278/Lab9$ touch Lab9.cpp
[hilgerbj@os1:~/cse278/Lab9$ nano Lab9.cpp
[hilgerbj@os1:~/cse278/Lab9$ g++ Lab9.cpp
[hilgerbj@os1:~/cse278/Lab9$ objdump -S a.out

a.out:     file format elf64-x86-64


Disassembly of section .init:

00000000000004b8 <_init>:
 4b8:   48 83 ec 08             sub    $0x8,%rsp
 4bc:   48 8b 05 25 0b 20 00    mov    0x200b25(%rip),%rax        # 200fe8 <__gmon_start__>
 4c3:   48 85 c0                test   %rax,%rax
 4c6:   74 02                   je     4ca <_init+0x12>
 4c8:   ff d0                   callq  *%rax
 4ca:   48 83 c4 08             add    $0x8,%rsp
 4ce:   c3                      retq

Disassembly of section .plt:

00000000000004d0 <.plt>:
 4d0:   ff 35 f2 0a 20 00       pushq  0x200af2(%rip)        # 200fc8 <_GLOBAL_OFFSET_TABLE_+0x8>
 4d6:   ff 25 f4 0a 20 00       jmpq   *0x200af4(%rip)        # 200fd0 <_GLOBAL_OFFSET_TABLE_+0x10>
 4dc:   0f 1f 40 00             nopl   0x0(%rax)

Disassembly of section .plt.got:

00000000000004e0 <__cxa_finalize@plt>:
 4e0:   ff 25 12 0b 20 00       jmpq   *0x200b12(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 4e6:   66 90                   xchg   %ax,%ax

Disassembly of section .text:

00000000000004f0 <_start>:
 4f0:   31 ed                   xor    %ebp,%ebp
 4f2:   49 89 d1                mov    %rdx,%r9
 4f5:   5e                      pop    %rsi
 4f6:   48 89 e2                mov    %rsp,%rdx
 4f9:   48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
 4fd:   50                      push   %rax
 4fe:   54                      push   %rsp
 4ff:   4c 8d 05 9a 01 00 00    lea    0x19a(%rip),%r8        # 6a0 <__libc_csu_fini>
 506:   48 8d 0d 23 01 00 00    lea    0x123(%rip),%rcx        # 630 <__libc_csu_init>
 50d:   48 8d 3d 02 01 00 00    lea    0x102(%rip),%rdi        # 616 <main>
 514:   ff 15 c6 0a 20 00       callq  *0x200ac6(%rip)        # 200fe0 <__libc_start_main@GLIBC_2.2.5>
 51a:   f4                      hlt
 51b:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

0000000000000520 <deregister_tm_clones>:
 520:   48 8d 3d e9 0a 20 00    lea    0x200ae9(%rip),%rdi        # 201010 <__TMC_END__>
 527:   55                      push   %rbp
 528:   48 8d 05 e1 0a 20 00    lea    0x200ae1(%rip),%rax        # 201010 <__TMC_END__>
 52f:   48 39 f8                cmp    %rdi,%rax
 532:   48 89 e5                mov    %rsp,%rbp
 535:   74 19                   je     550 <deregister_tm_clones+0x30>
 537:   48 8b 05 9a 0a 20 00    mov    0x200a9a(%rip),%rax        # 200fd8 <_ITM_deregisterTMCloneTable>
 53e:   48 85 c0                test   %rax,%rax
 541:   74 0d                   je     550 <deregister_tm_clones+0x30>
 543:   5d                      pop    %rbp
 544:   ff e0                   jmpq   *%rax
 546:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 54d:   00 00 00
 550:   5d                      pop    %rbp
 551:   c3                      retq
 552:   0f 1f 40 00             nopl   0x0(%rax)
 556:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 55d:   00 00 00
```



```
0000000000000560 <register_tm_clones>:
 560:   48 8d 3d a9 0a 20 00    lea    0x200aa9(%rip),%rdi        # 201010 <__TMC_END__>
 567:   48 8d 35 a2 0a 20 00    lea    0x200aa2(%rip),%rsi        # 201010 <__TMC_END__>
 56e:   55                      push   %rbp
 56f:   48 29 fe                sub    %rdi,%rsi
 572:   48 89 e5                mov    %rsp,%rbp
 575:   48 c1 fe 03             sar    $0x3,%rsi
 579:   48 89 f0                mov    %rsi,%rax
 57c:   48 c1 e8 3f             shr    $0x3f,%rax
 580:   48 01 c6                add    %rax,%rsi
 583:   48 d1 fe                sar    %rsi
 586:   74 18                   je     5a0 <register_tm_clones+0x40>
 588:   48 8b 05 61 0a 20 00    mov    0x200a61(%rip),%rax        # 200ff0 <_ITM_registerTMCloneTable>
 58f:   48 85 c0                test   %rax,%rax
 592:   74 0c                   je     5a0 <register_tm_clones+0x40>
 594:   5d                      pop    %rbp
 595:   ff e0                   jmpq   *%rax
 597:   66 0f 1f 84 00 00 00    nopw   0x0(%rax,%rax,1)
 59e:   00 00
 5a0:   5d                      pop    %rbp
 5a1:   c3                      retq
 5a2:   0f 1f 40 00             nopl   0x0(%rax)
 5a6:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 5ad:   00 00 00

00000000000005b0 <__do_global_dtors_aux>:
 5b0:   80 3d 59 0a 20 00 00    cmpb   $0x0,0x200a59(%rip)        # 201010 <__TMC_END__>
 5b7:   75 2f                   jne    5e8 <__do_global_dtors_aux+0x38>
 5b9:   48 83 3d 37 0a 20 00    cmpq   $0x0,0x200a37(%rip)        # 200ff8 <__cxa_finalize@GLIBC_2.2.5>
 5c0:   00
 5c1:   55                      push   %rbp
 5c2:   48 89 e5                mov    %rsp,%rbp
 5c5:   74 0c                   je     5d3 <__do_global_dtors_aux+0x23>
 5c7:   48 8b 3d 3a 0a 20 00    mov    0x200a3a(%rip),%rdi        # 201008 <__dso_handle>
 5ce:   e8 0d ff ff ff          callq  4e0 <__cxa_finalize@plt>
 5d3:   e8 48 ff ff ff          callq  520 <deregister_tm_clones>
 5d8:   c6 05 31 0a 20 00 01    movb   $0x1,0x200a31(%rip)        # 201010 <__TMC_END__>
 5df:   5d                      pop    %rbp
 5e0:   c3                      retq
 5e1:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)
 5e8:   f3 c3                   repz retq
 5ea:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)

00000000000005f0 <frame_dummy>:
 5f0:   55                      push   %rbp
 5f1:   48 89 e5                mov    %rsp,%rbp
 5f4:   5d                      pop    %rbp
 5f5:   e9 66 ff ff ff          jmpq   560 <register_tm_clones>

00000000000005fa <_Z4testv>:
 5fa:   55                      push   %rbp
 5fb:   48 89 e5                mov    %rsp,%rbp
 5fe:   c7 45 f8 01 00 00 00    movl   $0x1,-0x8(%rbp)
 605:   c7 45 fc 02 00 00 00    movl   $0x2,-0x4(%rbp)
 60c:   8b 55 f8                mov    -0x8(%rbp),%edx
 60f:   8b 45 fc                mov    -0x4(%rbp),%eax
 612:   01 d0                   add    %edx,%eax
 614:   5d                      pop    %rbp
 615:   c3                      retq

0000000000000616 <main>:
 616:   55                      push   %rbp
 617:   48 89 e5                mov    %rsp,%rbp
 61a:   e8 db ff ff ff          callq  5fa <_Z4testv>
 61f:   b8 00 00 00 00          mov    $0x0,%eax
 624:   5d                      pop    %rbp
 625:   c3                      retq
 626:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 62d:   00 00 00
000000000000630 <__libc_csu_init>:
630:   41 57                   push   %r15
632:   41 56                   push   %r14
634:   49 89 d7                mov    %rdx,%r15
637:   41 55                   push   %r13
639:   41 54                   push   %r12
63b:   4c 8d 25 ae 07 20 00    lea    0x2007ae(%rip),%r12        # 200df0 <__frame_dummy_init_array_entry>
642:   55                      push   %rbp
643:   48 8d 2d ae 07 20 00    lea    0x2007ae(%rip),%rbp        # 200df8 <__init_array_end>
64a:   53                      push   %rbx
64b:   41 89 fd                mov    %edi,%r13d
64e:   49 89 f6                mov    %rsi,%r14
651:   4c 29 e5                sub    %r12,%rbp
654:   48 83 ec 08             sub    $0x8,%rsp
658:   48 c1 fd 03             sar    $0x3,%rbp
65c:   e8 57 fe ff ff          callq  4b8 <_init>
661:   48 85 ed                test   %rbp,%rbp
664:   74 20                   je     686 <__libc_csu_init+0x56>
666:   31 db                   xor    %ebx,%ebx
668:   0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
66f:   00
670:   4c 89 fa                mov    %r15,%rdx
673:   4c 89 f6                mov    %r14,%rsi
676:   44 89 ef                mov    %r13d,%edi
679:   41 ff 14 dc             callq  *(%r12,%rbx,8)
67d:   48 83 c3 01             add    $0x1,%rbx
681:   48 39 dd                cmp    %rbx,%rbp
684:   75 ea                   jne    670 <__libc_csu_init+0x40>
686:   48 83 c4 08             add    $0x8,%rsp
68a:   5b                      pop    %rbx
68b:   5d                      pop    %rbp
68c:   41 5c                   pop    %r12
68e:   41 5d                   pop    %r13
690:   41 5e                   pop    %r14
692:   41 5f                   pop    %r15
694:   c3                      retq
695:   90                      nop
696:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
69d:   00 00 00

0000000000000006a0 <__libc_csu_fini>:
6a0:   f3 c3                   repz retq

isassembly of section .fini:

0000000000000006a4 <_fini>:
6a4:   48 83 ec 08             sub    $0x8,%rsp
6a8:   48 83 c4 08             add    $0x8,%rsp
6ac:   c3                      retq
ilgerbj@os1:~/cse278/Lab9$ 
```

2. Now modify the code as follows:

```
        int test( int val1, int val2)
        {
                return val1 + val2;
        }

        int main (void)
        {
            auto ret = test (42, 42);
        }
```

a. Again from the resulting binary, report the generated assembly language code for relevant for the **main()** function and **test()** function

b. Issue the following command:

readelf –SW a.out

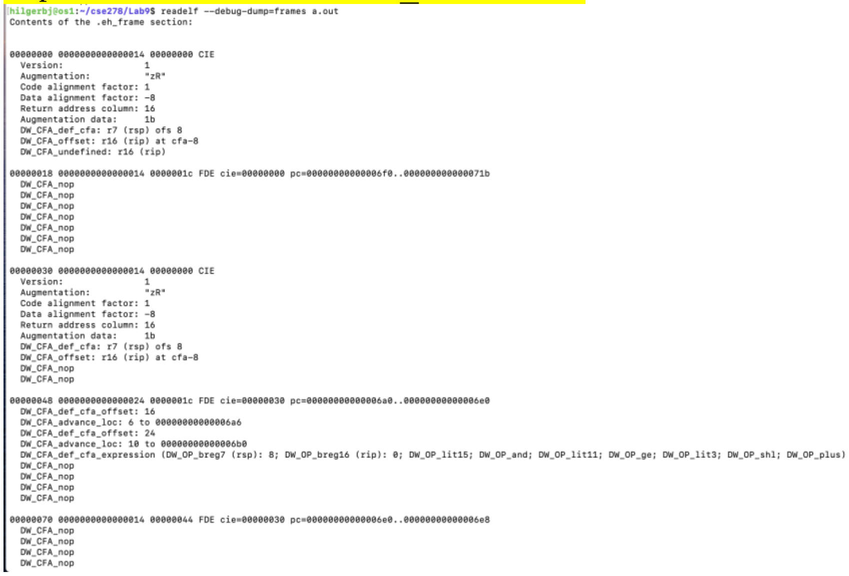
How many sections reported there?

There are 28 sections reported

c. To know more details of readelf, issue the command
man readelf

d. Issue the following command:
readelf --debug-dump=frames a.out

Report the contents of the `.eh_frame` table



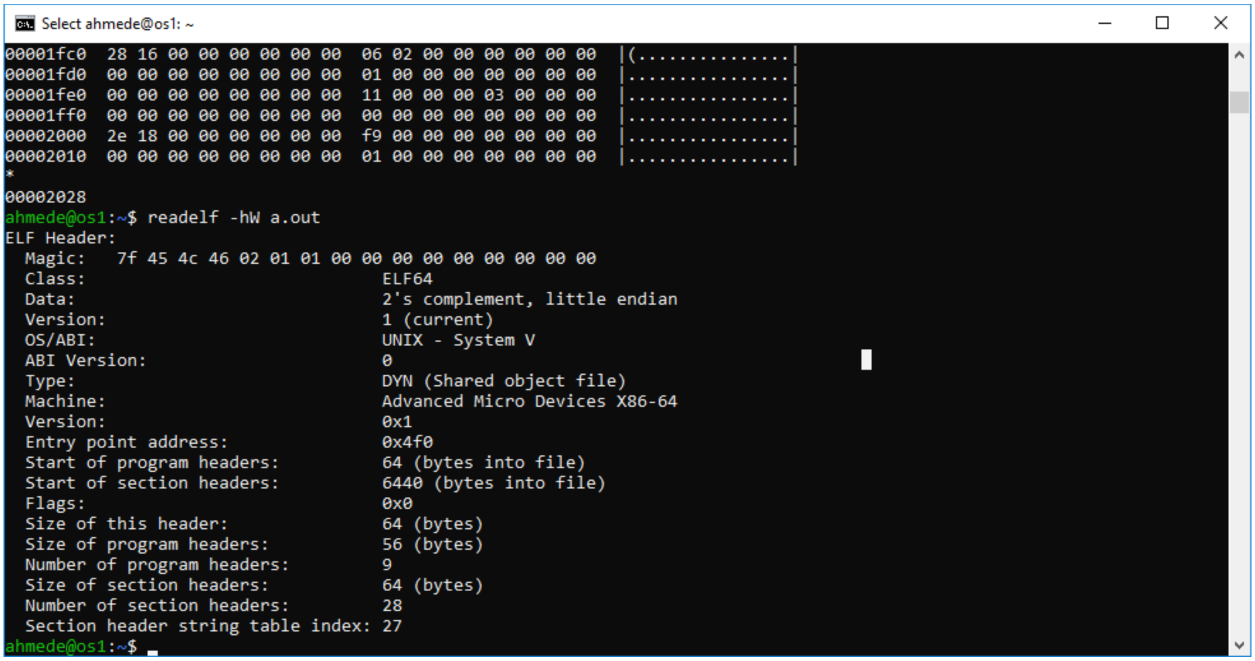
```
hilgerbj@os1:~/cse278/Lab9$ readelf --debug-dump=frames a.out
Contents of the .eh_frame section:


00000000 0000000000000014 00000000 CIE
  Version:               1
  Augmentation:          "zR"
  Code alignment factor: 1
  Data alignment factor: -8
  Return address column: 16
  Augmentation data:     1b
  DW_CFA_def_cfa: r7 (rsp) ofs 8
  DW_CFA_offset: r16 (rip) at cfa-8
  DW_CFA_undefined: r16 (rip)

00000018 0000000000000014 0000001c FDE cie=00000000 pc=00000000000006f0..000000000000071b
  DW_CFA_nop
  DW_CFA_nop
  DW_CFA_nop
  DW_CFA_nop
  DW_CFA_nop
  DW_CFA_nop
  DW_CFA_nop

00000030 0000000000000014 00000000 CIE
  Version:               1
  Augmentation:          "zR"
  Code alignment factor: 1
  Data alignment factor: -8
  Return address column: 16
  Augmentation data:     1b
  DW_CFA_def_cfa: r7 (rsp) ofs 8
  DW_CFA_offset: r16 (rip) at cfa-8
  DW_CFA_nop
  DW_CFA_nop

00000048 0000000000000024 0000001c FDE cie=00000030 pc=00000000000006a0..00000000000006e0
  DW_CFA_def_cfa_offset: 16
  DW_CFA_advance_loc: 6 to 00000000000006a6
  DW_CFA_def_cfa_offset: 24
  DW_CFA_advance_loc: 10 to 00000000000006b0
  DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8; DW_OP_breg16 (rip): 0; DW_OP_lit15; DW_OP_and; DW_OP_lit11; DW_OP_ge; DW_OP_lit3; DW_OP_shl; DW_OP_plus)
  DW_CFA_nop
  DW_CFA_nop
  DW_CFA_nop
  DW_CFA_nop

00000070 0000000000000014 00000044 FDE cie=00000030 pc=00000000000006e0..00000000000006e8
  DW_CFA_nop
  DW_CFA_nop
  DW_CFA_nop
  DW_CFA_nop
```

The .eh_frame contains sections of CIE (Common Information Entry) and FDE (Frame Description Entry) that handle certain parts of exceptions. Every CIE is followed by one or more FDE in the output and each memory address comes one after the other.

e. Look at the *hexdump* of the resulting a.out ELF by issuing the following command:
hexdump –C a.out

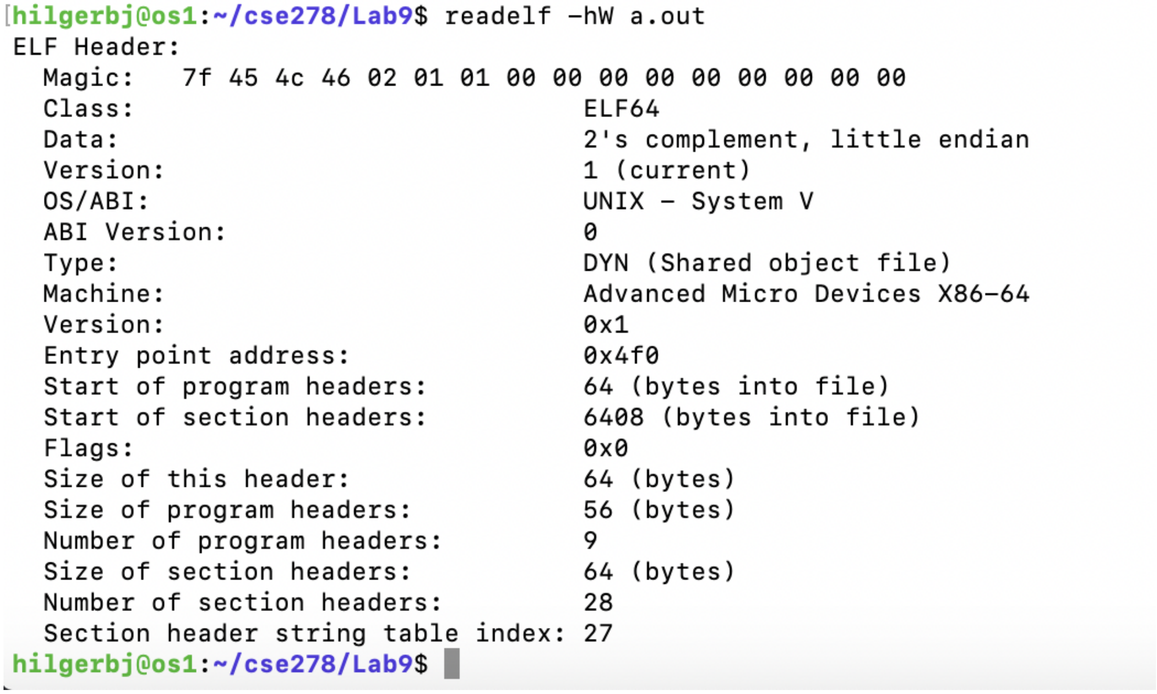
Every ELF file begins with the hex number 0x7F, and continues with the ELF string.

Issue the following command to view the ELF file's header:
readelf –hW a.out
You should see a result like this:



```
00001fc0  28 16 00 00 00 00 00 00  06 02 00 00 00 00 00 00  |(...............|
00001fd0  00 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00  |................|
00001fe0  00 00 00 00 00 00 00 00  11 00 00 00 03 00 00 00  |................|
00001ff0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00002000  2e 18 00 00 00 00 00 00  f9 00 00 00 00 00 00 00  |................|
00002010  00 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00  |................|
*
00002028
ahmede@os1:~$ readelf -hW a.out
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:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x4f0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          6440 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         9
  Size of section headers:           64 (bytes)
  Number of section headers:         28
  Section header string table index: 27
ahmede@os1:~$
```

Attach the screen shot what you get from previous command.

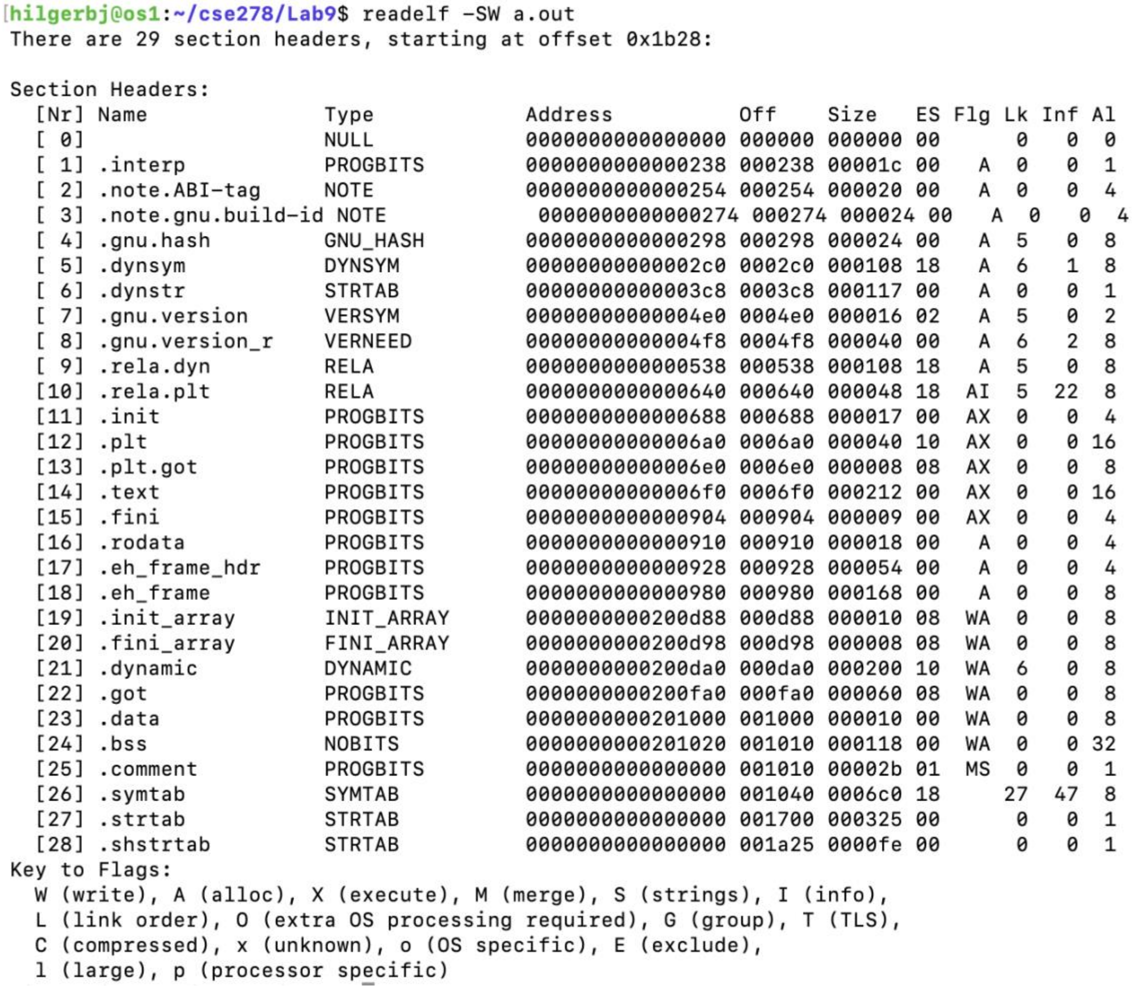


```
[hilgerbj@os1:~/cse278/Lab9$ readelf -hW a.out
 ELF Header:
   Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
   Class:                             ELF64
   Data:                              2's complement, little endian
   Version:                           1 (current)
   OS/ABI:                            UNIX - System V
   ABI Version:                       0
   Type:                              DYN (Shared object file)
   Machine:                           Advanced Micro Devices X86-64
   Version:                           0x1
   Entry point address:               0x4f0
   Start of program headers:          64 (bytes into file)
   Start of section headers:          6408 (bytes into file)
   Flags:                             0x0
   Size of this header:               64 (bytes)
   Size of program headers:           56 (bytes)
   Number of program headers:         9
   Size of section headers:           64 (bytes)
   Number of section headers:         28
   Section header string table index: 27
 hilgerbj@os1:~/cse278/Lab9$
```

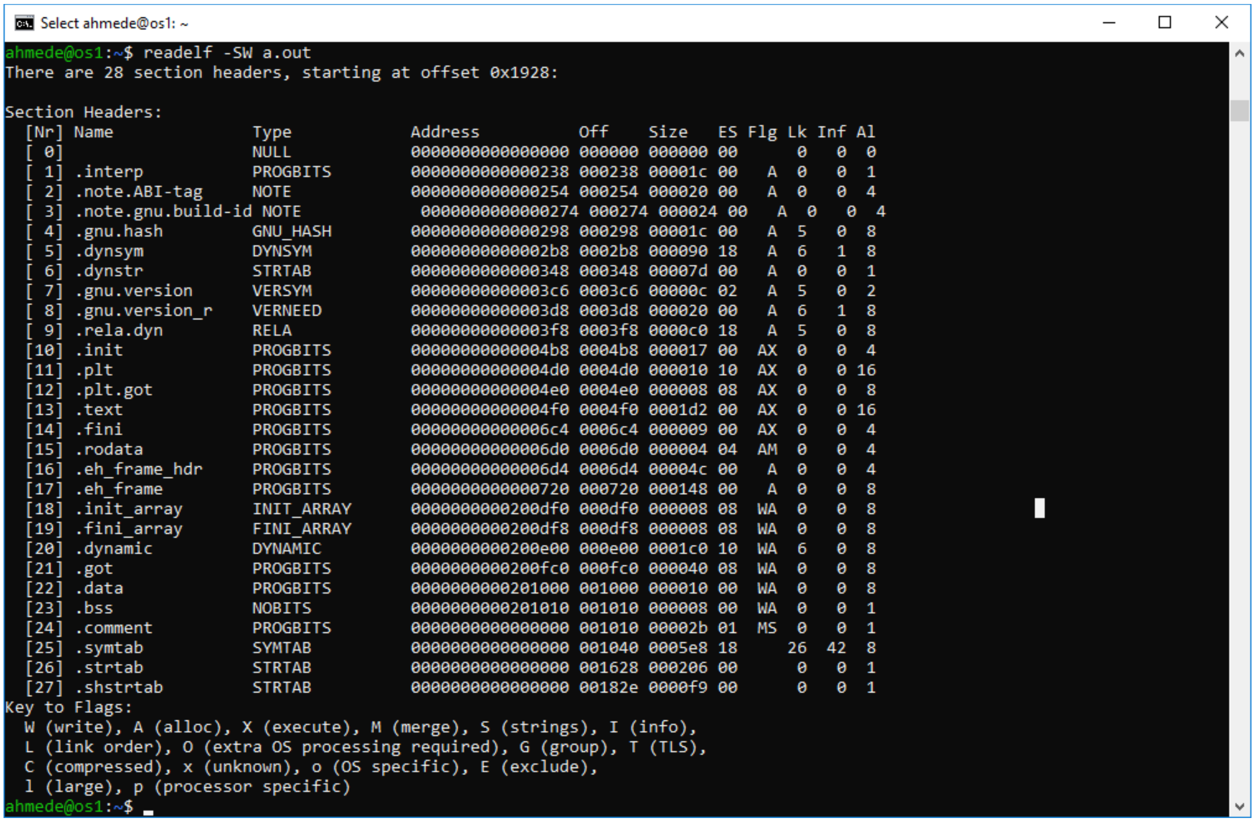
f. ELF sections

To see a list of all the sections, use the following command:
readelf –SW a.out

This will result in something like the following output:



```
[hilgerbj@os1:~/cse278/Lab9$ readelf -SW a.out
There are 29 section headers, starting at offset 0x1b28:

Section Headers:
  [Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        0000000000000238 000238 00001c 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            0000000000000254 000254 000020 00   A  0   0  4
  [ 3] .note.gnu.build-id NOTE            0000000000000274 000274 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        0000000000000298 000298 000024 00   A  5   0  8
  [ 5] .dynsym           DYNSYM          00000000000002c0 0002c0 000108 18   A  6   1  8
  [ 6] .dynstr           STRTAB          00000000000003c8 0003c8 000117 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          00000000000004e0 0004e0 000016 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         00000000000004f8 0004f8 000040 00   A  6   2  8
  [ 9] .rela.dyn         RELA            0000000000000538 000538 000108 18   A  5   0  8
  [10] .rela.plt         RELA            0000000000000640 000640 000048 18  AI  5  22  8
  [11] .init             PROGBITS        0000000000000688 000688 000017 00  AX  0   0  4
  [12] .plt              PROGBITS        00000000000006a0 0006a0 000040 10  AX  0   0 16
  [13] .plt.got          PROGBITS        00000000000006e0 0006e0 000008 08  AX  0   0  8
  [14] .text             PROGBITS        00000000000006f0 0006f0 000212 00  AX  0   0 16
  [15] .fini             PROGBITS        0000000000000904 000904 000009 00  AX  0   0  4
  [16] .rodata           PROGBITS        0000000000000910 000910 000018 00   A  0   0  4
  [17] .eh_frame_hdr     PROGBITS        0000000000000928 000928 000054 00   A  0   0  4
  [18] .eh_frame         PROGBITS        0000000000000980 000980 000168 00   A  0   0  8
  [19] .init_array       INIT_ARRAY      0000000000200d88 000d88 000010 08  WA  0   0  8
  [20] .fini_array       FINI_ARRAY      0000000000200d98 000d98 000008 08  WA  0   0  8
  [21] .dynamic          DYNAMIC         0000000000200da0 000da0 000200 10  WA  6   0  8
  [22] .got              PROGBITS        0000000000200fa0 000fa0 000060 08  WA  0   0  8
  [23] .data             PROGBITS        0000000000201000 001000 000010 00  WA  0   0  8
  [24] .bss              NOBITS          0000000000201020 001010 000118 00  WA  0   0 32
  [25] .comment          PROGBITS        0000000000000000 001010 00002b 01  MS  0   0  1
  [26] .symtab           SYMTAB          0000000000000000 001040 0006c0 18     27  47  8
  [27] .strtab           STRTAB          0000000000000000 001700 000325 00      0   0  1
  [28] .shstrtab         STRTAB          0000000000000000 001a25 0000fe 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  l (large), p (processor specific)
```



```
ahmede@os1:~$ readelf -SW a.out
There are 28 section headers, starting at offset 0x1928:

Section Headers:
  [Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        0000000000000238 000238 00001c 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            0000000000000254 000254 000020 00   A  0   0  4
  [ 3] .note.gnu.build-id NOTE            0000000000000274 000274 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        0000000000000298 000298 00001c 00   A  5   0  8
  [ 5] .dynsym           DYNSYM          00000000000002b8 0002b8 000090 18   A  6   1  8
  [ 6] .dynstr           STRTAB          0000000000000348 000348 00007d 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          00000000000003c6 0003c6 00000c 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         00000000000003d8 0003d8 000020 00   A  6   1  8
  [ 9] .rela.dyn         RELA            00000000000003f8 0003f8 0000c0 18   A  5   0  8
  [10] .init             PROGBITS        00000000000004b8 0004b8 000017 00  AX  0   0  4
  [11] .plt              PROGBITS        00000000000004d0 0004d0 000010 10  AX  0   0 16
  [12] .plt.got          PROGBITS        00000000000004e0 0004e0 000008 08  AX  0   0  8
  [13] .text             PROGBITS        00000000000004f0 0004f0 0001d2 00  AX  0   0 16
  [14] .fini             PROGBITS        00000000000006c4 0006c4 000009 00  AX  0   0  4
  [15] .rodata           PROGBITS        00000000000006d0 0006d0 000004 04  AM  0   0  4
  [16] .eh_frame_hdr     PROGBITS        00000000000006d4 0006d4 00004c 00   A  0   0  4
  [17] .eh_frame         PROGBITS        0000000000000720 000720 000148 00   A  0   0  8
  [18] .init_array       INIT_ARRAY      0000000000200df0 000df0 000008 08  WA  0   0  8
  [19] .fini_array       FINI_ARRAY      0000000000200df8 000df8 000008 08  WA  0   0  8
  [20] .dynamic          DYNAMIC         0000000000200e00 000e00 0001c0 10  WA  6   0  8
  [21] .got              PROGBITS        0000000000200fc0 000fc0 000040 08  WA  0   0  8
  [22] .data             PROGBITS        0000000000201000 001000 000010 00  WA  0   0  8
  [23] .bss              NOBITS          0000000000201010 001010 000008 00  WA  0   0  1
  [24] .comment          PROGBITS        0000000000000000 001010 00002b 01  MS  0   0  1
  [25] .symtab           SYMTAB          0000000000000000 001040 0005e8 18     26  42  8
  [26] .strtab           STRTAB          0000000000000000 001628 000206 00      0   0  1
  [27] .shstrtab         STRTAB          0000000000000000 00182e 0000f9 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  l (large), p (processor specific)
ahmede@os1:~$ _
```

g. What you get from the above command, explain the following sections:

- eh_frame/.eh_frame_hdr

  The eh_frame and .eh_frame_hdr are in charge of handling exceptions and provide tables in how to unwind the stack. According to the report above, it is the PROGBITS type, which means it's a section that contains either initialized data and instructions or instructions only.
  .eh_frame_hdr has a memory location of 0000000000000928
  offset: 000928
  size: 000054 (84 bits)
  .eh_frame has a memory location of 0000000000000980
  Offset: 000980
  Size: 000168 (360 bits)
  0000000000000980 000980 000168

- .init_array/.fini_array/.init/.fini

  The .init_array/.fini_array/.init/.fini sections are in charge of the initialization anf termination functionality. According to the report, the .init_array is of type INIT_ARRAY, which means that this section contains an array of pointers to initializations functions. Then the .fini_array is of type FINI_ARRAY, which means that this section contains an array of pointers to termination functions. The .init and. .fini are both of type PROGBITS, which means they only contained already initialized data and instructions or instructions only.

.init_array has a memory location of 0000000000200d88

Offset: 000d88

Size: 000010

.finit_array has a memory location of 0000000000200d98 Offset: 000d98

Size: 000008

.init has a memory location of 0000000000000688

Offset: 000688

Size: 000017

.fini has a memory location of 0000000000000904

Offset: 000904

Size: 00000

- .dynsym
  The .dynsym section contains the symbol table necessary to support dynamic linking. According to the report, the .dynsym has a memory location of 00000000000002c0

  Offset: 0002c0

  Size: 000108

3. Recompile the Lab9.cpp, this time using –v flag

g++ Lab9.cpp –v

From the report generated, identify the **PIE** (Position Independent Executable)

A PIE is an executable that can be run from anywhere in the primary memory without worrying about the absolute memory address. This means that now that the file has been configured with PIE as shown by the flags in the report, it can be run anywhere in the primary memory and still function properly.

There are some notable sections that should be pointed out:

- .text (contains most of the code associated with the program)

- .data (contains global variables initialized other than 0)
- .bss (global variables that should be initialized to 0)

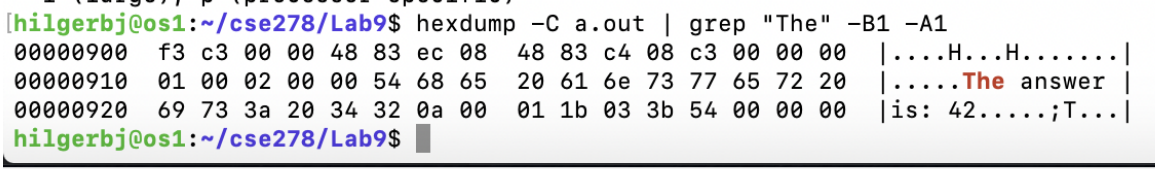
4. a. Now, modify the Lab9.cpp code, just to print one line of text as: "The answer is: 42\n". This time don't call any of the test() function from main().

Recompile the code

b. Issue the following command:

hexdump –C a.out | grep "The" -B1 –A1

Attach the screen shot what you see from above command.



```
[hilgerbj@os1:~/cse278/Lab9$ hexdump -C a.out | grep "The" -B1 -A1
00000900  f3 c3 00 00 48 83 ec 08  48 83 c4 08 c3 00 00 00  |....H...H.......|
00000910  01 00 02 00 00 54 68 65  20 61 6e 73 77 65 72 20  |.....The answer |
00000920  69 73 3a 20 34 32 0a 00  01 1b 03 3b 54 00 00 00  |is: 42.....;T...|
hilgerbj@os1:~/cse278/Lab9$
```

c. Answer the following questions

- What is a pipe in Linux? How we represent pipe?

We represent the pipe with the "|" character and a pipe is a form of redirecting data from one command to another. In the instance above, we are redirecting the output from the hexdump command to the grep command, which then the grep filters the data and then displays that to the console.

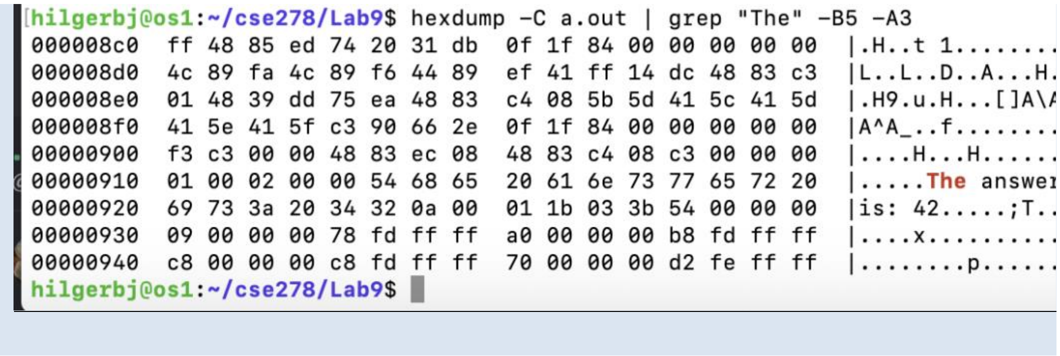
- Elaborate the meaning of the command grep.

Grep stands for the Global regular expression print and searches files for a specified pattern of characters and then returns the memory addresses that contain that specified pattern.

- Make an *educated guess* what B and A might refer to in the above command.

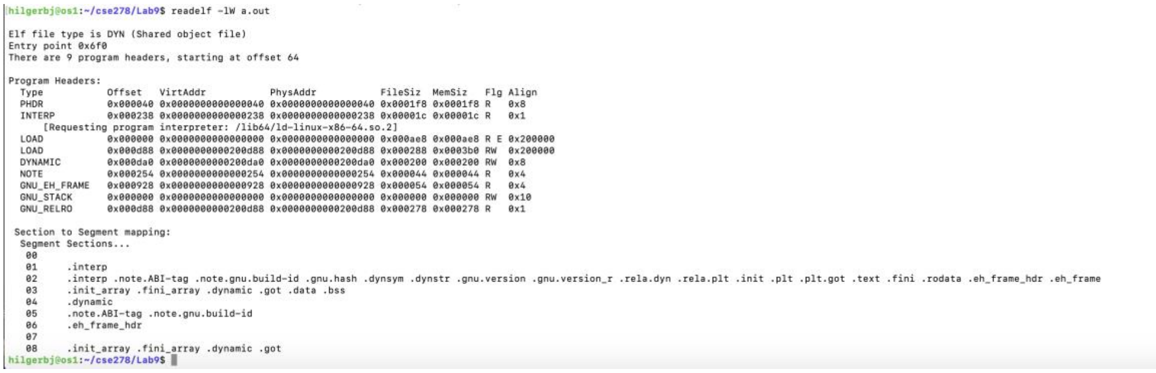
  The B1 flag displays the contents of one (hence B1) memory address found after the main address found using grep
  The A1 flag displays the contents of one (hence A1) memory address after the main address found using grep

- Use different B and A values for the above, then attach another screen shot what you get



```
hilgerbj@os1:~/cse278/Lab9$ hexdump -C a.out | grep "The" -B5 -A3
000008c0  ff 48 85 ed 74 20 31 db  0f 1f 84 00 00 00 00 00  |.H..t 1........
000008d0  4c 89 fa 4c 89 f6 44 89  ef 41 ff 14 dc 48 83 c3  |L..L..D..A...H.
000008e0  01 48 39 dd 75 ea 48 83  c4 08 5b 5d 41 5c 41 5d  |.H9.u.H...[]A\A
000008f0  41 5e 41 5f c3 90 66 2e  0f 1f 84 00 00 00 00 00  |A^A_..f........
00000900  f3 c3 00 00 48 83 ec 08  48 83 c4 08 c3 00 00 00  |....H...H......
00000910  01 00 02 00 00 54 68 65  20 61 6e 73 77 65 72 20  |.....The answer
00000920  69 73 3a 20 34 32 0a 00  01 1b 03 3b 54 00 00 00  |is: 42.....;T..
00000930  09 00 00 00 78 fd ff ff  a0 00 00 00 b8 fd ff ff  |....x..........
00000940  c8 00 00 00 c8 fd ff ff  70 00 00 00 d2 fe ff ff  |........p......
hilgerbj@os1:~/cse278/Lab9$
```

5. Issue the following command to see the loadable components:
   readelf –lW a.out
   Attach the screen shot what you see from above command



```
hilgerbj@os1:~/cse278/Lab9$ readelf -lW a.out

Elf file type is DYN (Shared object file)
Entry point 0x6f0
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
  PHDR           0x000040 0x0000000000000040 0x0000000000000040 0x0001f8 0x0001f8 R   0x8
  INTERP         0x000238 0x0000000000000238 0x0000000000000238 0x00001c 0x00001c R   0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x000000 0x0000000000000000 0x0000000000000000 0x000ae8 0x000ae8 R E 0x200000
  LOAD           0x000d88 0x0000000000200d88 0x0000000000200d88 0x000288 0x0003b0 RW  0x200000
  DYNAMIC        0x000da0 0x0000000000200da0 0x0000000000200da0 0x000200 0x000200 RW  0x8
  NOTE           0x000254 0x0000000000000254 0x0000000000000254 0x000044 0x000044 R   0x4
  GNU_EH_FRAME   0x000928 0x0000000000000928 0x0000000000000928 0x000054 0x000054 R   0x4
  GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x10
  GNU_RELRO      0x000d88 0x0000000000200d88 0x0000000000200d88 0x000278 0x000278 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .plt.got .text .fini .rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .dynamic .got .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .dynamic .got
hilgerbj@os1:~/cse278/Lab9$
```

PART B: Debugger (gdb)

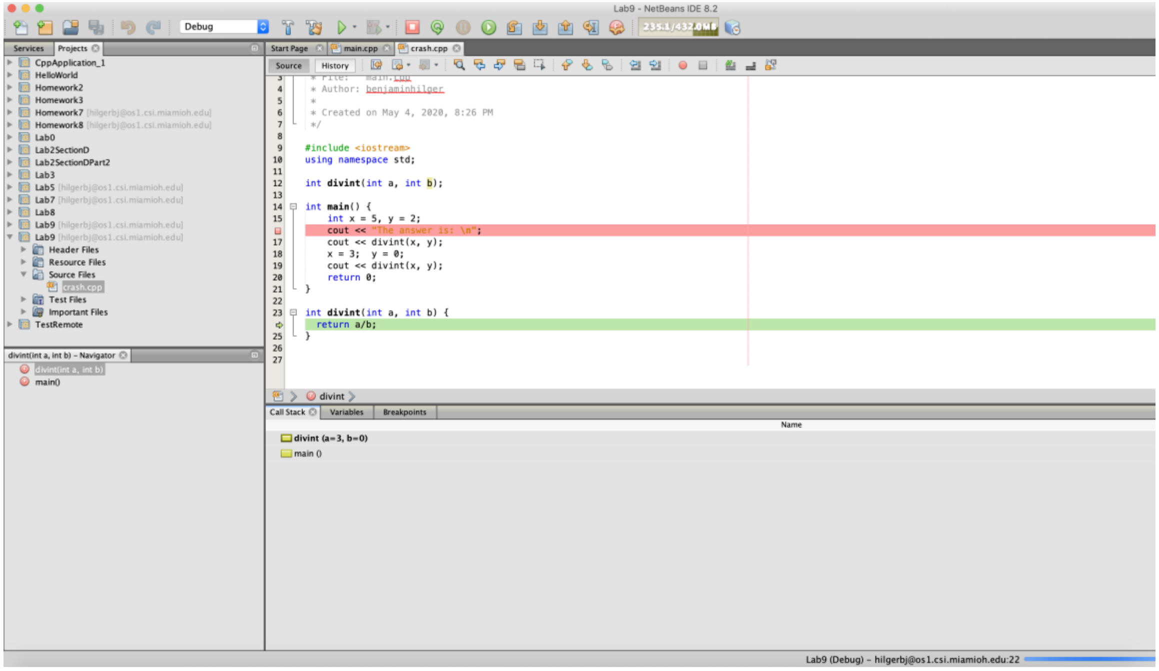
Goals

- How to write more efficient code
- Where to look when hard-to-find bugs arise.

Note: You may use the supplied **crash.cpp** in os1.csi.miamioh.edu (compile using -g switch) to experiment with all the commands that are available in gdb, *alternatively*, you may use your own sample code to experiment and should use NetBeans8.2 built-in gdb debugger. Here are the operations you should try:

1. Try setting and removing breakpoints
2. Step line-by-line in a method (may be the main()) and observe changes in a variable
3. Try navigating the stack frames in the debugger (observe how the call stack looks)

4. Finally, make a screenshot of your whole NetBeans window showing:
   a. The current line of code
   b. The stack trace tab
   c. The variables tab



```
 * File:   main.cpp
 * Author: benjaminhilger
 *
 * Created on May 4, 2020, 8:26 PM
 */

#include <iostream>
using namespace std;

int divint(int a, int b);

int main() {
    int x = 5, y = 2;
    cout << "The answer is: \n";
    cout << divint(x, y);
    x = 3;  y = 0;
    cout << divint(x, y);
    return 0;
}

int divint(int a, int b) {
  return a/b;
}
```

```
divint (a=3, b=0)
main ()
```

Lab9 (Debug) – hilgerbj@os1.csi.miamioh.edu:22

- Starting gdb
  - gdb nameOfExecutable
- Running a Program
  - (gdb) run
- Stack Trace
  - (gdb) backtrace
- Examining Variables
  - (gdb) print x
- Listing the Program
  - (gdb) list
- Setting Breakpoints
  - (gdb) help breakpoint

From command shell, issue
$ gdb –help
It will show all the commands usage

The same help can be found when running gdb already:

```
$gdb
(gdb) help
```

To quit from debugger, use the command q

```
(gdb) q
```

Command	**Purpose**
q /quit	Quit from gdb
L	List
l myfunction	
b main	Puts a breakpoint at the main
break 5	Set a breakpoint at line 5 and run the program
Run	
b N	Puts breakpoint at line N
r /run	Start script
Where	
Up	
p x	Print variable
Cont	Continue
commnads	End with a line saying just "end"

Submission

- No late assignments will be accepted!
- This work is to be done individually
- The submission file will be saved with the name ***Lab9_yourMUID.pdf***
- Assignment is due by Mon/Tue May 4/5 during Lab time.
- On or before the due time, drop the *electronic copy* of your work in the *canvas*
- Don't forget to Turn in the files!   Lab9_yourMUID.pdf & Lab9_yourMUID*.cpp