Q1: Objdump 1

a)

main.o

main.o: file format elf64-x86-64

Disassembly of section .text:

```
0000000000000000 <main>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   b8 00 00 00 00          mov    $0x0,%eax
   d:   e8 00 00 00 00          callq  12 <main+0x12>
                    e: R_X86_64_PLT32       swap-0x4
  12:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 18 <main+0x18>
                    14: R_X86_64_PC32       buf
  18:   8b 05 00 00 00 00       mov    0x0(%rip),%eax        # 1e <main+0x1e>
                    1a: R_X86_64_PC32       buf-0x4
  1e:   89 c6                   mov    %eax,%esi
  20:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # 27 <main+0x27>
                    23: R_X86_64_PC32       .rodata-0x4
  27:   b8 00 00 00 00          mov    $0x0,%eax
  2c:   e8 00 00 00 00          callq  31 <main+0x31>
                    2d: R_X86_64_PLT32      printf-0x4
  31:   b8 00 00 00 00          mov    $0x0,%eax
  36:   5d                      pop    %rbp
  37:   c3                      retq
```

Disassembly of section .data:

```
0000000000000000 <buf>:
   0:   72 00                   jb     2 <buf+0x2>
   2:   00 00                   add    %al,(%rax)
   4:   56                      push   %rsi
   5:   00 00                   add    %al,(%rax)
        ...
```

Disassembly of section .rodata:

```
0000000000000000 <.rodata>:
```

```
   0:   62                  (bad)
   1:   75 66               jne    69 <main+0x69>
   3:   5b                  pop    %rbx
   4:   30 5d 3d              xor    %bl,0x3d(%rbp)
   7:   20 25 64 20 62 75       and    %ah,0x75622064(%rip)        # 75622071
<main+0x75622071>
   d:   66 5b               pop    %bx
   f:   31 5d 3d              xor    %ebx,0x3d(%rbp)
  12:   20                  .byte 0x20
  13:   25                  .byte 0x25
  14:   64 0a 00               or     %fs:(%rax),%al

Disassembly of section .comment:

0000000000000000 <.comment>:
   0:   00 47 43               add    %al,0x43(%rdi)
   3:   43 3a 20               rex.XB cmp (%r8),%spl
   6:   28 55 62               sub    %dl,0x62(%rbp)
   9:   75 6e                  jne    79 <main+0x79>
   b:   74 75                  je     82 <main+0x82>
   d:   20 39                  and    %bh,(%rcx)
   f:   2e 32 2e               xor    %cs:(%rsi),%ch
  12:   31 2d 39 75 62 75       xor    %ebp,0x75627539(%rip)        # 75627551
<main+0x75627551>
  18:   6e                  outsb  %ds:(%rsi),(%dx)
  19:   74 75                  je     90 <main+0x90>
  1b:   32 29                  xor    (%rcx),%ch
  1d:   20 39                  and    %bh,(%rcx)
  1f:   2e 32 2e               xor    %cs:(%rsi),%ch
  22:   31 20                  xor    %esp,(%rax)
  24:   32 30                  xor    (%rax),%dh
  26:   31 39                  xor    %edi,(%rcx)
  28:   31 30                  xor    %esi,(%rax)
  2a:   30 38                  xor    %bh,(%rax)
        ...

Disassembly of section .note.gnu.property:

0000000000000000 <.note.gnu.property>:
   0:   04 00                  add    $0x0,%al
   2:   00 00                  add    %al,(%rax)
   4:   10 00                  adc    %al,(%rax)
   6:   00 00                  add    %al,(%rax)
   8:   05 00 00 00 47            add    $0x47000000,%eax
```

```
  d:   4e 55                rex.WRX push %rbp
  f:   00 02                add    %al,(%rdx)
 11:   00 00                add    %al,(%rax)
 13:   c0 04 00 00             rolb   $0x0,(%rax,%rax,1)
 17:   00 03                add    %al,(%rbx)
 19:   00 00                add    %al,(%rax)
 1b:   00 00                add    %al,(%rax)
 1d:   00 00                add    %al,(%rax)
       ...

Disassembly of section .eh_frame:

0000000000000000 <.eh_frame>:
   0:   14 00                adc    $0x0,%al
   2:   00 00                add    %al,(%rax)
   4:   00 00                add    %al,(%rax)
   6:   00 00                add    %al,(%rax)
   8:   01 7a 52                add    %edi,0x52(%rdx)
   b:   00 01                add    %al,(%rcx)
   d:   78 10                js     1f <.eh_frame+0x1f>
   f:   01 1b                add    %ebx,(%rbx)
  11:   0c 07                or     $0x7,%al
  13:   08 90 01 00 00 1c       or     %dl,0x1c000001(%rax)
  19:   00 00                add    %al,(%rax)
  1b:   00 1c 00                add    %bl,(%rax,%rax,1)
  1e:   00 00                add    %al,(%rax)
  20:   00 00                add    %al,(%rax)
                    20: R_X86_64_PC32       .text
  22:   00 00                add    %al,(%rax)
  24:   38 00                cmp    %al,(%rax)
  26:   00 00                add    %al,(%rax)
  28:   00 45 0e                add    %al,0xe(%rbp)
  2b:   10 86 02 43 0d 06       adc    %al,0x60d4302(%rsi)
  31:   6f                  outsl  %ds:(%rsi),(%dx)
  32:   0c 07                or     $0x7,%al
  34:   08 00                or     %al,(%rax)
```

swap.o

```
swap.o:     file format elf64-x86-64


Disassembly of section .text:
```

```
0000000000000000 <swap>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax        # f <swap+0xf>
                        b: R_X86_64_PC32        buf
   f:   48 89 05 00 00 00 00    mov    %rax,0x0(%rip)        # 16 <swap+0x16>
                        12: R_X86_64_PC32       bufp1-0x4
  16:   48 8b 05 00 00 00 00    mov    0x0(%rip),%rax        # 1d <swap+0x1d>
                        19: R_X86_64_PC32       bufp0-0x4
  1d:   8b 00                   mov    (%rax),%eax
  1f:   89 05 00 00 00 00       mov    %eax,0x0(%rip)        # 25 <swap+0x25>
                        21: R_X86_64_PC32       .data-0x4
  25:   48 8b 15 00 00 00 00    mov    0x0(%rip),%rdx        # 2c <swap+0x2c>
                        28: R_X86_64_PC32       bufp1-0x4
  2c:   48 8b 05 00 00 00 00    mov    0x0(%rip),%rax        # 33 <swap+0x33>
                        2f: R_X86_64_PC32       bufp0-0x4
  33:   8b 12                   mov    (%rdx),%edx
  35:   89 10                   mov    %edx,(%rax)
  37:   48 8b 05 00 00 00 00    mov    0x0(%rip),%rax        # 3e <swap+0x3e>
                        3a: R_X86_64_PC32       bufp1-0x4
  3e:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 44 <swap+0x44>
                        40: R_X86_64_PC32       .data-0x4
  44:   89 10                   mov    %edx,(%rax)
  46:   90                      nop
  47:   5d                      pop    %rbp
  48:   c3                      retq

Disassembly of section .data:

0000000000000000 <temp.1915>:
   0:   64 00 00                add    %al,%fs:(%rax)
        ...

Disassembly of section .data.rel:

0000000000000000 <bufp0>:
        ...
                        0: R_X86_64_64  buf

Disassembly of section .comment:

0000000000000000 <.comment>:
   0:   00 47 43                add    %al,0x43(%rdi)
```

```
   3:   43 3a 20              rex.XB cmp (%r8),%spl
   6:   28 55 62              sub    %dl,0x62(%rbp)
   9:   75 6e                 jne    79 <swap+0x79>
   b:   74 75                 je     82 <swap+0x82>
   d:   20 39                 and    %bh,(%rcx)
   f:   2e 32 2e              xor    %cs:(%rsi),%ch
  12:   31 2d 39 75 62 75     xor    %ebp,0x75627539(%rip)        # 75627551
<swap+0x75627551>
  18:   6e                   outsb  %ds:(%rsi),(%dx)
  19:   74 75                je     90 <swap+0x90>
  1b:   32 29                xor    (%rcx),%ch
  1d:   20 39                and    %bh,(%rcx)
  1f:   2e 32 2e             xor    %cs:(%rsi),%ch
  22:   31 20                xor    %esp,(%rax)
  24:   32 30                xor    (%rax),%dh
  26:   31 39                xor    %edi,(%rcx)
  28:   31 30                xor    %esi,(%rax)
  2a:   30 38                xor    %bh,(%rax)
        ...

Disassembly of section .note.gnu.property:

0000000000000000 <.note.gnu.property>:
   0:   04 00                 add    $0x0,%al
   2:   00 00                 add    %al,(%rax)
   4:   10 00                 adc    %al,(%rax)
   6:   00 00                 add    %al,(%rax)
   8:   05 00 00 00 47        add    $0x47000000,%eax
   d:   4e 55                 rex.WRX push %rbp
   f:   00 02                 add    %al,(%rdx)
  11:   00 00                 add    %al,(%rax)
  13:   c0 04 00 00           rolb   $0x0,(%rax,%rax,1)
  17:   00 03                 add    %al,(%rbx)
  19:   00 00                 add    %al,(%rax)
  1b:   00 00                 add    %al,(%rax)
  1d:   00 00                 add    %al,(%rax)
        ...

Disassembly of section .eh_frame:

0000000000000000 <.eh_frame>:
   0:   14 00                 adc    $0x0,%al
   2:   00 00                 add    %al,(%rax)
   4:   00 00                 add    %al,(%rax)
```

```
  6:   00 00                 add    %al,(%rax)
  8:   01 7a 52               add    %edi,0x52(%rdx)
  b:   00 01                 add    %al,(%rcx)
  d:   78 10                 js     1f <.eh_frame+0x1f>
  f:   01 1b                 add    %ebx,(%rbx)
 11:   0c 07                 or     $0x7,%al
 13:   08 90 01 00 00 1c     or     %dl,0x1c000001(%rax)
 19:   00 00                 add    %al,(%rax)
 1b:   00 1c 00              add    %bl,(%rax,%rax,1)
 1e:   00 00                 add    %al,(%rax)
 20:   00 00                 add    %al,(%rax)
                 20: R_X86_64_PC32       .text
 22:   00 00                 add    %al,(%rax)
 24:   49 00 00              rex.WB add %al,(%r8)
 27:   00 00                 add    %al,(%rax)
 29:   45 0e                 rex.RB (bad)
 2b:   10 86 02 43 0d 06     adc    %al,0x60d4302(%rsi)
 31:   02 40 0c              add    0xc(%rax),%al
 34:   07                   (bad)
 35:   08 00                 or     %al,(%rax)
```

other.o

other.o: file format elf64-x86-64

Disassembly of section .text:

```
0000000000000000 <f>:
  0:   f3 0f 1e fa            endbr64
  4:   55                    push   %rbp
  5:   48 89 e5               mov    %rsp,%rbp
  8:   b8 00 00 00 00         mov    $0x0,%eax
  d:   e8 03 00 00 00         callq  15 <sf>
 12:   90                    nop
 13:   5d                    pop    %rbp
 14:   c3                    retq

0000000000000015 <sf>:
 15:   f3 0f 1e fa            endbr64
 19:   55                    push   %rbp
 1a:   48 89 e5               mov    %rsp,%rbp
 1d:   c7 05 00 00 00 00 03    movl   $0x3,0x0(%rip)        # 27 <sf+0x12>
 24:   00 00 00
                 1f: R_X86_64_PC32       buf-0x8
```

```
  27:   c7 05 00 00 00 00 04    movl   $0x4,0x0(%rip)        # 31 <sf+0x1c>
  2e:   00 00 00
                29: R_X86_64_PC32       buf-0x4
  31:   90                    nop
  32:   5d                    pop    %rbp
  33:   c3                    retq

Disassembly of section .comment:

0000000000000000 <.comment>:
   0:   00 47 43               add    %al,0x43(%rdi)
   3:   43 3a 20               rex.XB cmp (%r8),%spl
   6:   28 55 62               sub    %dl,0x62(%rbp)
   9:   75 6e                  jne    79 <sf+0x64>
   b:   74 75                  je     82 <sf+0x6d>
   d:   20 39                  and    %bh,(%rcx)
   f:   2e 32 2e               xor    %cs:(%rsi),%ch
  12:   31 2d 39 75 62 75      xor    %ebp,0x75627539(%rip)        # 75627551 <sf+0x7562753c>
  18:   6e                    outsb  %ds:(%rsi),(%dx)
  19:   74 75                  je     90 <sf+0x7b>
  1b:   32 29                  xor    (%rcx),%ch
  1d:   20 39                  and    %bh,(%rcx)
  1f:   2e 32 2e               xor    %cs:(%rsi),%ch
  22:   31 20                  xor    %esp,(%rax)
  24:   32 30                  xor    (%rax),%dh
  26:   31 39                  xor    %edi,(%rcx)
  28:   31 30                  xor    %esi,(%rax)
  2a:   30 38                  xor    %bh,(%rax)
        ...

Disassembly of section .note.gnu.property:

0000000000000000 <.note.gnu.property>:
   0:   04 00                  add    $0x0,%al
   2:   00 00                  add    %al,(%rax)
   4:   10 00                  adc    %al,(%rax)
   6:   00 00                  add    %al,(%rax)
   8:   05 00 00 00 47         add    $0x47000000,%eax
   d:   4e 55                  rex.WRX push %rbp
   f:   00 02                  add    %al,(%rdx)
  11:   00 00                  add    %al,(%rax)
  13:   c0 04 00 00            rolb   $0x0,(%rax,%rax,1)
  17:   00 03                  add    %al,(%rbx)
  19:   00 00                  add    %al,(%rax)
```

```
  1b:   00 00                 add    %al,(%rax)
  1d:   00 00                 add    %al,(%rax)
        ...

Disassembly of section .eh_frame:

0000000000000000 <.eh_frame>:
   0:   14 00                 adc    $0x0,%al
   2:   00 00                 add    %al,(%rax)
   4:   00 00                 add    %al,(%rax)
   6:   00 00                 add    %al,(%rax)
   8:   01 7a 52               add    %edi,0x52(%rdx)
   b:   00 01                 add    %al,(%rcx)
   d:   78 10                 js     1f <.eh_frame+0x1f>
   f:   01 1b                 add    %ebx,(%rbx)
  11:   0c 07                 or     $0x7,%al
  13:   08 90 01 00 00 1c       or     %dl,0x1c000001(%rax)
  19:   00 00                 add    %al,(%rax)
  1b:   00 1c 00               add    %bl,(%rax,%rax,1)
  1e:   00 00                 add    %al,(%rax)
  20:   00 00                 add    %al,(%rax)
                    20: R_X86_64_PC32       .text
  22:   00 00                 add    %al,(%rax)
  24:   15 00 00 00 00          adc    $0x0,%eax
  29:   45 0e                 rex.RB (bad)
  2b:   10 86 02 43 0d 06       adc    %al,0x60d4302(%rsi)
  31:   4c 0c 07               rex.WR or $0x7,%al
  34:   08 00                 or     %al,(%rax)
  36:   00 00                 add    %al,(%rax)
  38:   1c 00                 sbb    $0x0,%al
  3a:   00 00                 add    %al,(%rax)
  3c:   3c 00                 cmp    $0x0,%al
  3e:   00 00                 add    %al,(%rax)
  40:   00 00                 add    %al,(%rax)
                    40: R_X86_64_PC32       .text+0x15
  42:   00 00                 add    %al,(%rax)
  44:   1f                    (bad)
  45:   00 00                 add    %al,(%rax)
  47:   00 00                 add    %al,(%rax)
  49:   45 0e                 rex.RB (bad)
  4b:   10 86 02 43 0d 06       adc    %al,0x60d4302(%rsi)
  51:   56                    push   %rsi
  52:   0c 07                 or     $0x7,%al
  54:   08 00                 or     %al,(%rax)
```

a) **main.o :**

.text

1. 1e: R_X86_64_PLT32 swap-0x4
2. 14: R_X86_64_PC32 buf
3. 1a: R_X86_64_PC32 buf-0x4
4. 23: R_X86_64_PC32 .rodata-0x4
5. 2d: R_X86_64_PLT32 printf-0x4

.eh_frame

1. 20: R_X86_64_PC32 .text+0x0

swap.o :

.text

1. b: R_X86_64_PC32 buf
2. 12: R_X86_64_PC32 bufp1-0x4
3. 19: R_X86_64_PC32 bufp0-0x4
4. 21: R_X86_64_PC32 .data-0x4
5. 28: R_X86_64_PC32 bufp1-0x4
6. 2f: R_X86_64_PC32 bufp0-0x4
7. 3a: R_X86_64_PC32 bufp1-0x4
8. 40: R_X86_64_PC32 .data-0x4

.data.rel

1. 0: R_X86_64_64 buf

.eh_frame

1. 20: R_X86_64_PC32 .text+0x0

other.o :

.text

1. 1f: R_X86_64_PC32 buf-0x8
2. 29: R_X86_64_PC32 buf-0x4

.eh_frame

1. 20: R_X86_64_PC32 .text+0x0
2. 40: R_X86_64_PC32 .text+0x15

b) **Final address:**

a.out :

main.o :

.text

e: R_X86_64_PLT32 swap-0x4	0000000000001181 <swap> (.text)
14: R_X86_64_PC32 buf	0000000000004010 <buf> (.data)

1a: R_X86_64_PC32 buf-0x4	0000000000004014 <buf> (.data)
23: R_X86_64_PC32 .rodata-0x4	0000000000002004 (.rodata)
2d: R_X86_64_PLT32 printf-0x4	0000000000001050 <printf@plt>

.eh_frame

20: R_X86_64_PC32 .text+0x0	0000000000001060

swap.o :
.text

b: R_X86_64_PC32 buf	0000000000004014 <buf> (.data)
12: R_X86_64_PC32 bufp1-0x4	0000000000004030 <bufp1> (.data)
19: R_X86_64_PC32 bufp0-0x4	0000000000004020 <bufp0> (.data)
21: R_X86_64_PC32 .data-0x4	0000000000004018 <temp.1915>
28: R_X86_64_PC32 bufp1-0x4	0000000000004030 <bufp1> (.data)
2f: R_X86_64_PC32 bufp0-0x4	0000000000004020 <bufp0> (.data)
3a: R_X86_64_PC32 bufp1-0x4	0000000000004030 <bufp1> (.data)
40: R_X86_64_PC32 .data-0x4	0000000000004018 <temp.1915>

.data.rel

b: R_X86_64_PC32 buf	0000000000004014 <buf> (.data)

.eh_frame

20: R_X86_64_PC32 .text+0x0	0000000000001060
40: R_X86_64_PC32 .text+0x15	0000000000001075

other.o :

.text

1f: R_X86_64_PC32 buf-0x8	0000000000004010 <buf> (.data)
29: R_X86_64_PC32 buf-0x4	0000000000004014 <buf> (.data)

.eh_frame

20: R_X86_64_PC32 .text+0x0	0000000000001060

Q2: Objdump 2

(a)

a) int main()
```
401ce5:  f3 0f 1e fa          endbr64
401ce9:  55                   push   %rbp
401cea:  48 89 e5             mov    %rsp,%rbp
```
b) int a=1, b=1;
```
401ced:  c7 45 f8 01 00 00 00    movl   $0x1,-0x8(%rbp) ----------(a = 1)
401cf4:  c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp) ----------(b = 1)
```

c) while(a<=10)
```
401cfb:  eb 0e                jmp    401d0b <main+0x26>
```
d) b=b*a;
```
401cfd:  8b 45 fc             mov    -0x4(%rbp),%eax
401d00:  0f af 45 f8          imul   -0x8(%rbp),%eax
401d04:  89 45 fc             mov    %eax,-0x4(%rbp)
```
e) a++;
```
401d07:  83 45 f8 01          addl   $0x1,-0x8(%rbp)
```
f) while(a<=10)
```
401d0b:  83 7d f8 0a          cmpl   $0xa,-0x8(%rbp)
401d0f:  7e ec                jle    401cfd <main+0x18>
```
g) return b;
```
401d11:  8b 45 fc             mov    -0x4(%rbp),%eax
401d14:  5d                   pop    %rbp
401d15:  c3                   retq
401d16:  66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
401d1d:  00 00 00
```

(b) int a: -0x8(%rbp) , int b: -0x4(%rbp)

Q3: Objdump 3

(a)

a) struct data {
   int sum;
   int b[5];
  };
  int main()

```
 401ce5:    f3 0f 1e fa          endbr64
 401ce9:    55                   push   %rbp
 401cea:    48 89 e5             mov    %rsp,%rbp
 401ced:    48 83 ec 20          sub    $0x20,%rsp
 401cf1:    64 48 8b 04 25 28 00 mov    %fs:0x28,%rax
 401cf8:    00 00
 401cfa:    48 89 45 f8          mov    %rax,-0x8(%rbp)
 401cfe:    31 c0                xor    %eax,%eax
```

b) struct data rec1;
  rec1.sum=0;

```
 401d00:    c7 45 e0 00 00 00 00 movl   $0x0,-0x20(%rbp)
```

c) rec1.b[0]=2;
  ⇒ ⇒

```
 401d07:    c7 45 e4 02 00 00 00 movl   $0x2,-0x1c(%rbp)
```

d) rec1.sum=rec1.sum+rec1.b[0];

```
 401d0e:    8b 55 e0             mov    -0x20(%rbp),%edx
 401d11:    8b 45 e4             mov    -0x1c(%rbp),%eax
 401d14:    01 d0                add    %edx,%eax
 401d16:    89 45 e0             mov    %eax,-0x20(%rbp)
```

e) return rec1.sum;

```
 401d19:    8b 45 e0             mov    -0x20(%rbp),%eax
 401d1c:    48 8b 4d f8          mov    -0x8(%rbp),%rcx
 401d20:    64 48 33 0c 25 28 00 xor    %fs:0x28,%rcx
 401d27:    00 00
 401d29:    74 05                je     401d30 <main+0x4b>
 401d2b:    e8 d0 a0 04 00       callq  44be00 <__stack_chk_fail>
 401d30:    c9                   leaveq
 401d31:    c3                   retq
 401d32:    66 2e 0f 1f 84 00 00 nopw   %cs:0x0(%rax,%rax,1)
 401d39:    00 00 00
 401d3c:    0f 1f 40 00          nopl   0x0(%rax)
```

(b) rec1.sum : -0x20(%rbp)
  rec1.b : -0x1c(%rbp)

Q4: Static Linking

Part (a)
Here, Module a1 has function main() and Module a2 has uninitialized global variable main.
Thus, function main in a1 is the strong symbol and main in a2 is weak symbol.
In such cases the linker always chooses the strong symbol. **Thus, the use of the symbol main in module a2 will resolve to the declaration of main in module a1.**
Checking the symbol table shows function main as the Global Symbol.

⇒ readelf -s a.out
⇒ ⇒ 1441: 0000000000401ce5 56 FUNC GLOBAL DEFAULT 7 main

0000000000401ce5 is the address of the main function.
Objdump also shows that references to main are for the main function.
The output produced shows same values. (Address of main function)
⇒ 0x401ce5
⇒ 0x401ce5

Part (b)
Here the linker gives error as both module b1 and module b2 have strong symbols.
b1 has function main() (Strong) and b2 has global initialized variable (int main = 1;) which is also strong. **This violates the rule that there should exist only one strong symbol with same name.** Thus the linker returns an **error.**

Part (c)
Here we get two symbols for main. Module c1.c has the Global Symbol of main function and c2.c has Local Symbol static int main = 1.

⇒ readelf -s a.out
⇒ ⇒ 167: 00000000004be0f0 4 OBJECT LOCAL DEFAULT 21 main
⇒ ⇒ 1442: 0000000000401ce5 56 FUNC GLOBAL DEFAULT 7 main

The printf in c1.c can only reference the strong symbol (main function) because static variables and functions can be accessed only from their file thus the address printed is that of main function. Whereas in c2.c we have a local symbol (static int main) thus according to the scoping rules the printf statement prints the address of Local Symbol main.
Thus the output produced shows different addresses:
⇒ 0x401ce5
⇒ 0x4be0f0

The use of the symbol main in module c1 will resolve to the declaration of main in module c1 & the use of the symbol main in module c2 will resolve to the declaration of main in module c2.

Q5: Memory Layout

Symbol	Section	Region Number
main()	.text (Read Only)	5
f()	.text (Read Only)	5
malloc()	.plt.sec (Read Only)	5
printf()	.plt.sec (Read Only)	5
x	.data (Read/Write)	4
y	.data (Read/Write)	4
a	.bss (Read/Write)	4
k	.bss (Read/Write)	4
p	User Stack	1

*** p is not a symbol ??? ***

Q6: Link Issue!

a) Output :
⇒ 5 13
⇒ 13 5

Here we can see that printf in f() outputs swapped values of x and y, whereas printf in main() prints the correct values of x and y.
rec in module a1.c is Strong Symbol and that in module a2.c is Weak Symbol. Thus, the linker chooses struct rec of module a1.c.
We need to notice that struct in a2.c has y before and then x. Thus, references made to x in a2.c have address corresponding to that of y in struct rec in module a1.c.
The assembly code generated for a2.c mention location for rec.x as rec+0x4 and for rec.y as rec+0x0. But after linking rec refers to definition in module a1.c. Thus, rec+0x0

points to x and rec+0x4 points to y. Thus, the output produced is according to addresses of the assembly generated.

---

main:

```
printf(" %d %d\n", rec.x, rec.y);
   115b:      8b 15 b3 2e 00 00       mov    0x2eb3(%rip),%edx        # 4014 <rec+0x4>
   1161:      8b 05 a9 2e 00 00       mov    0x2ea9(%rip),%eax        # 4010 <rec>
   1167:      89 c6                   mov    %eax,%esi
   1169:      48 8d 3d 94 0e 00 00    lea    0xe94(%rip),%rdi        # 2004
<_IO_stdin_used+0x4>
   1170:      b8 00 00 00 00          mov    $0x0,%eax
   1175:      e8 d6 fe ff ff          callq  1050 <printf@plt>
```

f1:

```
   printf(" %d %d\n", rec.x, rec.y);
   1189:      8b 15 81 2e 00 00       mov    0x2e81(%rip),%edx        # 4010 <rec>
   118f:      8b 05 7f 2e 00 00       mov    0x2e7f(%rip),%eax        # 4014 <rec+0x4>
   1195:      89 c6                   mov    %eax,%esi
   1197:      48 8d 3d 6e 0e 00 00    lea    0xe6e(%rip),%rdi        # 200c
<_IO_stdin_used+0xc>
   119e:      b8 00 00 00 00          mov    $0x0,%eax
   11a3:      e8 a8 fe ff ff          callq  1050 <printf@plt>
```

We can see in objdump that the values in eax and edx are swapped for main() and f1().
----------That implies that referring to rec1.x in main() function points to location 0000000000004014 and rec1.y points to 0000000000004010.
Whereas referring to rec1.x in f1() function points to location 0000000000004010 and rec1.y in f1() function points to 0000000000004014.----------

---

b) Locations that need relocation:

test1.c

.text

1. e: R_X86_64_PLT32 f1-0x4
2. 14: R_X86_64_PC32 rec
3. 1a: R_X86_64_PC32 rec-0x4
4. 23: R_X86_64_PC32 .rodata-0x4
5. 2d: R_X86_64_PLT32 printf-0x4

.eh_frame

1. 20: R_X86_64_PC32 .text+0x0

test2.c

.text

1. a: R_X86_64_PC32 rec-0x4
2. 10: R_X86_64_PC32 rec

3. 19: R_X86_64_PC32 .rodata-0x4
4. 23: R_X86_64_PLT32 printf-0x4

.eh_frame

1. 20: R_X86_64_PC32 .text+0x0

c) Final addresses:

.text

main():

e: R_X86_64_PLT32 f1-0x4	0000000000001181 <f1>
14: R_X86_64_PC32 rec	0000000000004014 <rec+0x4>
1a: R_X86_64_PC32 rec-0x4	0000000000004010 <rec>
23: R_X86_64_PC32 .rodata-0x4	0000000000002004 <_IO_stdin_used+0x4>
2d: R_X86_64_PLT32 printf-0x4	1050 <printf@plt>

f1():

a: R_X86_64_PC32 rec-0x4	0000000000004010 <rec>
10: R_X86_64_PC32 rec	0000000000004014 <rec+0x4>
19: R_X86_64_PC32 .rodata-0x4	000000000000200c <_IO_stdin_used+0xc>
23: R_X86_64_PLT32 printf-0x4	1050 <printf@plt>

.eh_frame

20: R_X86_64_PC32 .text+0x0	0000000000001060

Q7: Static Linking - 2

a) We get error in line in module f2.c:

   ⇒ int z = a[3];

   This is because int a[] is defined as an extern variable in f2.c. Thus, it can not be used in that module.

b) The output observed for (5 == fn()) is 0. That is because d is no more 5 after statement: c = 100; in module f1.c
   In f1.c the assembly code generated is according to c being a double variable (8 bytes). But { int c = 58; } is the Strong Symbol. Thus the reference is made to this. Variable d occupies address right after c. Thus, when we execute c = 100 in f1.c, 8 bytes of data is written overriding d's actual value. Thus, d is not 5 anymore making e != 5. fn() doesn't return 5 and thus the output is 0.

c) Locations that need relocation:
   Address of locations that need relocation are mentioned with address relative to their section in the .o file.
   f1.c
   .text
   1. 11: R_X86_64_PC32 .rodata+0x4
   2. 19: R_X86_64_PC32 c-0x4
   3. 20: R_X86_64_PC32 c-0x4
   4. 2a: R_X86_64_PLT32 fn-0x4
   5. 41: R_X86_64_PC32 .rodata-0x4
   6. 4b: R_X86_64_PLT32 printf-0x4

   .data.rel.local ----------doubt
   1. 0: R_X86_64_64 a+0xc

   .data.rel ----------doubt
   1. 0: R_X86_64_64 b+0x0

   f2.c
   .text
   1. 0e: R_X86_64_PC32 d-0x4
   2. 1d: R_X86_64_PC32 .rodata-0x4
   3. 27: R_X86_64_PLT32 printf-0x4

   Final Addresses
   .text
   main()

Locations that need relocation	**Final address**
11: R_X86_64_PC32 .rodata+0x4	0000000000494010 <_IO_stdin_used+0x10>
19: R_X86_64_PC32 c-0x4	00000000004be118 <c>

20: R_X86_64_PC32 c-0x4	00000000004be118 <c>
2a: R_X86_64_PLT32 fn-0x4	0000000000401d40 <fn>
41: R_X86_64_PC32 .rodata-0x4	0000000000494008 <_IO_stdin_used+0x8>
4b: R_X86_64_PLT32 printf-0x4	0000000000410d40 <_IO_printf>

fn()

0e: R_X86_64_PC32 d-0x4	00000000004be11c <d>
1d: R_X86_64_PC32 .rodata-0x4	0000000000494018 <_IO_stdin_used+0x18>
27: R_X86_64_PLT32 printf-0x4	0000000000410d40 <_IO_printf>

.data.rel.local

0: R_X86_64_64 a+0xc	00000000004be0f0 <a> (.data)

.data.rel.

0: R_X86_64_64 b+0x0	00000000004be120 <b> (.data)

a. Symbols with relocation type R_X86_64_PC32 are relative to program counter.
    1. c
    1. d

b. Symbols with relocation type R_X86_64_PLT32 use the Program linkage table.
"In the case of a position-independent code, the call to a function be done through a table called the Procedure Linkage Table which is used by the dynamic linker at runtime. This type has been chosen in case we would link the nothing.o in a shared library and link the executable with this dynamic library. In the case all is statically linked, the linker will consider it will have to do the same job as if the relocation type was R_X86_64_PC32 relocation. "
Source:
https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-relocation-elf.html
Since, we assume static linking, relocation of the following symbols is relative to program counter.
    1. printf
    2. fn

c. Symbols with relocation type R_X86_64_64 are PC independent
    1. a
    2. b