Answer 1:
a.)

Main.o

.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_PLT32printf-0x4
  31:   b8 00 00 00 00          mov    $0x0,%eax
  36:   5d              pop    %rbp
  37:   c3              retq
```

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)
        ...
```

Swap.o

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
```

Other.o

```
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
```

All numbers in the tables below are in hexadecimal.

Main.o

Section	Offset(Initial)	Type	Symbol	Offset (Final)	Address(0x)
.text	e	R_X86_64_PLT32	swap	1156	11b5
	14	R_X86_64_PC32	buf[1]	115b	4014
	1a	R_X86_64_PC32	buf[0]	1161	4010
	23	R_X86_64_PC32	.rodata	1169	2004
	2d	R_X86_64_PLT32	printf	1175	1050

Swap.o

Section	Offset(Initial)	Type	Symbol	Offset (a.outl)	Address(0x)
.text	b	R_X86_64_PC32	buf[1]	11bd	4014
.text	12	R_X86_64_PC32	bufp1	11c4	4030
.text	19	R_X86_64_PC32	bufp0	11cb	4020

.text	21	R_X86_64_PC32	temp.1915temp	11d4	4018
.text	28	R_X86_64_PC32	bufp1	11da	4030
.text	2f	R_X86_64_PC32	bufp0	11e1	4020
.text	3a	R_X86_64_PC32	bufp1	11ec	4030
.text	40	R_X86_64_PC32	temp	11f3	4018

Other.o

Section	**Offset(Initial)**	**Type**	**Symbol**	**Offset (a.outl)**	**Address(0x)**
.text	1f	R_X86_64_PC32	buf[0]	11bd	4010
.text	29	R_X86_64_PC32	buf[1]	11c4	4014

Answer 2 :

```
 0:    f3 0f 1e fa             endbr64
 4:    55                      push   %rbp
 5:    48 89 e5                mov    %rsp,%rbp
 8:    c7 45 f8 01 00 00 00    movl   $0x1,-0x8(%rbp)
 f:    c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
16:    eb 0e                   jmp    26 <main+0x26>
18:    8b 45 fc                mov    -0x4(%rbp),%eax
1b:    0f af 45 f8             imul   -0x8(%rbp),%eax
1f:    89 45 fc                mov    %eax,-0x4(%rbp)
22:    83 45 f8 01             addl   $0x1,-0x8(%rbp)
26:    83 7d f8 0a             cmpl   $0xa,-0x8(%rbp)
2a:    7e ec                   jle    18 <main+0x18>
2c:    8b 45 fc                mov    -0x4(%rbp),%eax
2f:    5d                      pop    %rbp
30:    c3                      retq
```

Target Code	**Source Code**
push %rbp mov %rsp,%rbp	{
movl $0x1,-0x8(%rbp)	Int a = 1,
movl $0x1,-0x4(%rbp)	b=1
jmp 26 <main+0x26>	while(a<=10)

mov -0x4(%rbp),%eax
imul -0x8(%rbp),%eax
mov %eax,-0x4(%rbp)
	b=b*a;

addl $0x1,-0x8(%rbp)	a++;
cmpl $0xa,-0x8(%rbp) jle 18 <main+0x18>	while(a<=10){ }
mov -0x4(%rbp),%eax	return b

pop %rbp retq	}

Relative addresses of local variables w.r.t to %rbp:

a -0x8

b -0x4

Answer3:

Disassembly of section .text:

```
0000000000000000 <main>:
   0:   f3 0f 1e fa                endbr64
   4:   55                 push   %rbp
   5:   48 89 e5                   mov    %rsp,%rbp
   8:   48 83 ec 20                sub    $0x20,%rsp
   c:   64 48 8b 04 25 28 00       mov    %fs:0x28,%rax
  13:   00 00
  15:   48 89 45 f8                mov    %rax,-0x8(%rbp)
  19:   31 c0                      xor    %eax,%eax
  1b:   c7 45 e0 00 00 00 00       movl   $0x0,-0x20(%rbp)
  22:   c7 45 e4 02 00 00 00       movl   $0x2,-0x1c(%rbp)
  29:   8b 55 e0                   mov    -0x20(%rbp),%edx
  2c:   8b 45 e4                   mov    -0x1c(%rbp),%eax
  2f:   01 d0                      add    %edx,%eax
  31:   89 45 e0                   mov    %eax,-0x20(%rbp)
  34:   8b 45 e0                   mov    -0x20(%rbp),%eax
  37:   48 8b 4d f8                mov    -0x8(%rbp),%rcx
  3b:   64 48 33 0c 25 28 00       xor    %fs:0x28,%rcx
  42:   00 00
  44:   74 05                      je     4b <main+0x4b>
  46:   e8 00 00 00 00             callq  4b <main+0x4b>
  4b:   c9                 leaveq
  4c:   c3                 retq
```

Target Code	**Source Code**

endbr64
push %rbp
mov %rsp,%rbp
sub $0x20,%rsp
mov %fs:0x28,%rax
mov %rax,-0x8(%rbp)
xor %eax,%eax
	`struct data {`
	`  int sum;`
	`  int b[5];`
	`};`
	
	`int main()`
	`{`

movl $0x0,-0x20(%rbp)	`struct data rec1;`
	
	`  rec1.sum=0;`

movl $0x2,-0x1c(%rbp)	`rec1.b[0]=2;`

mov -0x20(%rbp),%edx
mov -0x1c(%rbp),%eax
add %edx,%eax
mov %eax,-0x20(%rbp)
	`rec1.sum=rec1.sum+rec1.b[0];`

mov -0x20(%rbp),%eax	`return rec1.sum;`
mov -0x8(%rbp),%rcx xor %fs:0x28,%rcx je 4b <main+0x4b> callq 4b <main+0x4b> leaveq retq	}

Relative address of local variable and parameters w.r.t %rbp

rec1.sum : -0x20

rec1.b : -0x1c

Answer4

a.)

Declaration of main in a1 is strong while in a2 is weak. (uninitialized global variable).
The use of the symbol main in module **a1** will resolve to the declaration of main in module **a1**.
The use of the symbol main in module **a2** will resolve to the declaration of main in module **a1**.

b.)
On linking b1 and b2, there will be an error because the main in both b1 and b2 contains main which is a strong symbol. Thus linker will give an error.

c.)
In c1 main is a strong symbol whereas c2 in main is a static variable. Thus on linking c1 with c2 no error occurs.
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**

Answer 5

Memory Segment	**Symbol Name**
User Stack	p
Region for Shared Libraries	printf,malloc
Heap	N.A
Read/Write Segment	x,y,a,k
Read-only Segment	main,f

Answer 6

a.) Output

5 13
13 5
Explained below
b.)

test1. o

Section : text

Offset	Type	Symbol
0xe	R_X86_64_PLT32	f1
0x14	R_X86_64_PC32	rec /rec.x
0x1a	R_X86_64_PC32	rec - 0x4/re .y
0x23	R_X86_64_PC32	odata
0x2d	R_X86_64_PLT32	printf

test2.0
Section: text

Offset	Type	Symbol
0xa	R_X86_64_PC32	rec-0x4/rec.x
0x10	R_X86_64_PC32	rec/rec.y
0x19	R_X86_64_PC32	.rodata
0x23	R_X86_64_PLT32	printf

c.)

Final Addresses

test1.o

Initial Offset	Type	Symbol	Final Offset	Address
0xe	R_X86_64_PLT32	f1	0x1156	1181
0x14	R_X86_64_PC32	rec+0x4/rec .x	0x115b	4014
0x1a	R_X86_64_PC32	rec/rec.y	0x1161	4010
0x23	R_X86_64_PC32	rodata	0x1169	2004
0x2d	R_X86_64_PLT32	printf	0x1175	1050

test2.0

Initial Offset	Type	Symbol	Final Offset	Address
0xa	R_X86_64_PC32	rec-0x4/rec.y	0x1189	4010
0x10	R_X86_64_PC32	rec/rec.x	0x118f	4014
0x19	R_X86_64_PC32	.r0xe6e(%rip),%rdi odata		0x1197
200c				
0x23	R_X86_64_PLT32	printf	0x11a3	1050

Reasoning for Part1:

When the object file of module 2 is generated, printf statement simply stores the relative addresses where rec.x and rec.y are present. Now since in module 2, x is declared after y, the address of x(rec-0x4) is 4 bytes after y(rec). When linking occurs, the value at addresses is overwritten by the rec struct of module 1 with {13, 5} as it is stronger than declaration of rec in module2. Hence the address y contains 13 and address x contains 5.thus at first 5,13 is printed. The next 13,5 is obvious.

Answer7 :

The following error comes:

```
f2.c:4:9: error: initializer element is not constant
    4 | int z = a[3];
      |         ^
```

Thus error come at the line int z = a[3]

In f2.c, the array a has been declared as a global, its initial size should be a constant since the compiler needs to be able to allocate size for it in the executable.

b.) The value returned by function f is not equal to 5. c is a weak symbol in module 1 and is resolved by c in module2. Since d was immediately declared after c, therefore they were stored consecutively in the same stack. Now since c was declared double in module1 which is of 8 bytes thus it overwrites c and d both due to which d now gives a garbage value, and therefore the 5==fn returning as 0.

c.)

Section is .text

File	**Symbol**	**Type of relocation**
f1.o	c	PC-Relative
f1.o	fn	Absolute

f1.o	printf	Absolute
f2.0	d	PC-Relative