CS240 Lab1
Ji Ma
ma438@purdue.edu

Q1

The nature of the a.out is the binary code compiled by gcc.
Steps:

1. Preprocess - process the # statement like include libs
2. Compile - Compile the main.c file into machine language.
3. Link - link the existed libs and remix it in the compiled code.

If I change the declared function head main to test, the gcc compiler will throw the error msg: undefined reference to 'main'. It won't generate an executable.

If you change the int to void, nothing will happen on the screen, but there will be no return specified in this function.

Q2

```
➜  lab1 git:(master) ✗ whereis stdio.h
stdio: /usr/include/stdio.h
```

The string that does not contain %d is literal, and the part %d is reserved for integer vars in this case.

if I run gcc with option -c, which means
-c Compile and assemble, but do not link

```
➜  v2 git:(master) ✗ gcc -c main.c
➜  v2 git:(master) ✗ ls
main.c  main.o
```

Main.o is the partial compiled file only for main.c, if we use gcc to merge main.o and myadd.o, we will get the finalized binary executable output.

Q3

& is a pointer in C, so when you are trying to assign value in scanf()

```
scanf("%d %d",&x,&y)
```

You are trying to assign integer values into the correct address.

If I remove & in front of vars.

```
scanf("%d %d",x,y)
```

```
➜  v3 git:(master) ✗ gcc main.c
main.c:9:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
main.c: In function ‘main’:
main.c:14:9: warning: format ‘%d’ expects argument of type ‘int *’, but argument 2 has
   scanf("%d %d",x,y);
         ^
main.c:14:9: warning: format ‘%d’ expects argument of type ‘int *’, but argument 3 has
```

```
➜  v3 git:(master) ✗ a.out
1 2
[1]    3565 segmentation fault  a.out
```

Q4

& refers to address, if you use &var in the printf(), you will print address.

```
// print result
  printf("%d plus %d = %d\n", &x, &y, &z);
```

```
➜  v3 git:(master) ✗ a.out
1 2
1148769420 plus 1148769424 = 1148769428
```

Q5

When we dealing with

```
// compute addition
  z = myadd(x,y);
```

Because the myadd function don't have to modify the variable directly from their address. So, we don't need &.

However, if we want to change the z value directly in the function, then we need to specify &z to let c know where is the address. The different between address and value is really important.

Here's some error I got from gcc

```
main.c:37:10: error: incompatible types when returning type ‘float *’ but ‘float’ was
 return c;
 main.c: In function ‘main’:
 main.c:17:17: error: incompatible type for argument 3 of ‘myadd’
    z = myadd(x,y,&z);
                  ^
 main.c:7:7: note: expected ‘float’ but argument is of type ‘float *’
  float myadd(float, float,float);
        ^
 main.c: At top level:
 main.c:30:7: error: conflicting types for ‘myadd’
  float myadd(float a, float b, float *c)
        ^
 main.c:7:7: note: previous declaration of ‘myadd’ was here
  float myadd(float, float,float);
```

So changed main.c with the knowledge showing right here:

1. For declaring parameter type for address, use xxxx *
2. Use * before var for any accepted address.
3. Use & before var for any modification on address.

Q6

V6 modualized the myadd and main in seperate files.

This is how you compile the v6 code. I figured out that if you modulized the function in different .c file, you have to pass all the file that you wrote to gcc to make it work.

```
➜  v6 git:(master) ✗ gcc main.c myadd.c
➜  v6 git:(master) ✗ a.out
1.321321 32.312321
result of 1.321321 plus 32.312321 is 33.63
```

V7 has removed the `float myadd(float, float);` in main function, instead we put this statement in myheader.h for reference(It's easier for we to manage, say we could put all the function link declaration in one file and include just only for once).

if I cange <> to ""

```
➜  v7 git:(master) ✗ gcc main.c
main.c:6:22: fatal error: myheader.h: No such file or directory
compilation terminated.
```

I would say that <> is for some system built in libs, "" is for user created function/lib in the same path.

Bonus problem

Basically, I added * for all the params in declaration. And add & infront of them when I call the function. Inside of the function, I use

```
float a_value = *a;
float b_value = *b;
```

to get the value from the address, and then we can add them together and give the value to the address of c.