Problem 1 (20 pts)

Q. What is the size of the output, i.e., file a.out, generated by gcc?

A. The size of a.out is 8168 bytes.

Q. What is the full pathname of a.out?

A. /homes/jang128/CS240_with_Park/lab01/lab1/v1/a.out

Q. What happens when you try to inspect the content of the file a.out by using the app cat, and why?

A. The terminal shows me a weird text with a lot of unreadable characters. Since the a.out file is a compiled code with the sequence of bytes, I thought `cat a.out` will show me the binary codes. However, the reason why `cat a.out` shows me the weird context instead of the binary codes is that the terminal recognizes the binary codes as the numbers and it tries to change those numbers as characters by using ASCII, UTF8, or UTF16. This is the reason why the terminal shows a weird context.

Q. The app gcc takes as input the file main.c and generates a new file a.out. From a logical perspective, how is the content of a.out related to the content of the file main.c?

A. Since the computer cannot understand our code, we need to change our code which is main.c to the binary code which is a.out. `gcc` which is one of the compilers of C language changes the main.c to a.out which is an executable program. There are mainly three steps for `gcc` to change the main.c to a.out.

The first process is preprocessing. In this step, the preprocessor reads the code and removes the comments from the code, and includes the headers which are located at the beginning of the files.

The second process is compiling. The compiler passes the filtered code in the preprocessor and generates the assembly code.

Last process is assembling. In the last step, the compiler changes the assembly code to the binary code which is only readable from the machine.

With these three steps, gcc creates a file called a.out.

a.out is a compiled code of the main.c which is only made up of binary codes and which is an executable file.

Q. In what sense is the C program contained in the file main.c not very useful?

A. The reason why the code in the main.c is not very useful is that the program does not print anything, and user cannot do anything with the program.

Problem 2 (20 pts)

Q. What steps have you undertaken to protect the content of your work in lab1/ under your working directory so that it is only accessible by you? Explain using the permission settings of folders lab1/ and v1/, and files main.c and a.out in v1/.

A. To protect the file or directory, we can set the permission so that only I can access to the file or folder by using the command `chmod`. The command to make the lab1/ be accessible only by me is and to grant the same permission to the subdirectories, we use `chmod -R 700 *`. By doing this way, we can see that permission to read, write, and execute is only granted to the owner.

To see the permission settings, we can use `ls -l` command.

The permission settings of folder lab1/ is `drwx--S---`.

The permission settings of folder v1/ is `drwx--S---`.
`drwx--S---` This means that this folder is a directory and only the owner can read, write or execute the directory. With the meaning of big 'S', this is called SetGid, but since we have not learned this in the class, TA said I don't have to explain this.
The permission settings of file main.c is `-rwx------`.
The permission settings of file a.out is `-rwx------`.
`-rwx------` This means that the file is not a directory and only the owner has permission to read, write and execute the file.

Problem 3 (20 pts)

Q. In your own words, explain what we expect to happen when we run the executable program contained in file a.out with the help of a shell app using
% a.out
A. When the user runs the `a.out`, the shell looks for the executable program. Once the shell found the executable program from the a.out file, it runs the function called fork(). This function gives the executable program a Process ID which lets shell execute the executable program to be processed. Once the executable program is terminated, the shell is getting the return value.

Q. Although using your own words, make the description technically meaningful. In the test cases discussed in class, the shell app used was /bin/tcsh. **Check what shell you are running (there are several variants) with the help of the ps command.** One of the apps we will code in a later lab assignment is a simplified shell app. You will find that the core structure of a shell app, stripped of myriad bells and whistles, is but a few lines of C code. It is an instance of one of the most practically important apps running on today's computing systems, called client/server apps, and one of the key applications of C programming.
A. The shell that I am using is zsh.

Problem 4 (20 pts)

An application software coded in C may be viewed as a collection of functions whose execution starts at the designated function main() which may call other functions that, in turn, may make further nested function calls.
Q. Given that main() is where supposedly execution begins, why is it that main() itself is declared to return a value of type int? Note that in v1/main.c the last statement of main() is "return 0".
A. There are two reasons why main() function is a type of int.
Firstly, if the type of main() is int, then this means that main() should return the integer. `return 0` conventionally means that the code is compiled successfully.
Second, the result of main() function passes to the exit() system call in UNIX systems. If 0 passes the exit(), then this is considered as exit success. However, if other numbers than 0 pass the exit(), then this is considered as exit failure.
Those are the reason why main() function returns a value of type int.

Q. What happens if you remove the type qualifier int in front of main() in v1/main.c and compile using gcc? Did compilation succeed? Explain.
A. The compilation fails if I remove the `int` in front of main() in v1/main.c and gives me this error message.
` warning: return type defaults to 'int' [-Wimplicit-int]`.

Since the default return type is int in C89, there should be `int` in front of the `main()`.

Q. What happens if you modify the spelling of main() to Main() and compile? Did compilation succeed? Explain.
A. The compilation fails and give me the error message like:
`/usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu/Scrt1.o: In function `_start':
(.text+0x20): undefined reference to `main'
collect2: error: ld returned 1 exit status`.
This means that there is no `main()` function in the code. Since the compiler differentiates the capital letters and the small letters, the compiler will not recognize the `Main()` as a `main()` function. For the compiler, it seems like there is no `main()` function in the program, but there should be the `main()` function to compile the code successfully. The reason why there should be `main()` function is that the `main()` is a start up itself to execute the program by the compiler.

Problem 5 (20 pts)

The version in directory v2/ is essentially the same as the one in v1/ except that printing of the subtraction result is facilitated by the libary function printf(). A library function is code that others have written so that we may utilize them in our own code through linking. printf() is part of the standard I/O library (a collection of functions aimed at facilitating input/output operations) and outputs to standard output which is, by default, the terminal (i.e., display or screen) associated with a Linux PC in our labs. If you access a lab machine remotely using secure shell, the terminal of your local machine acts as standard output. To use the standard I/O library, we must include the header file stdio.h using the #include C preprocessor directive.

Q. Find where the file stdio.h is located, i.e., its full pathname.
A. Stdio.h is located at `/usr/include/stdio.h`.

Q. How large is the file?
A. Stdio.h is 29,665 bytes.

Q. What are its protection bits, and what do they convey?
A. The protection bits of stdio.h means `-rw-r--r--`.
This means that stdio.h is a file, not a directory.
`rw-` means that the owner can only read and write the file, but not execute.
`r--` means that people who are in the same group as the owner only can read, but not write and execute.
The last `r--` means that other people who are not in the group and who are not the owner only can read, but not write and execute the file.
This protection code conveys that only the owner can write and read the file. Other people who are not an owner can just read the stdio.h file.

Q. Look into stdio.h and identify the line number where printf() is specified. Note that the function prototype of any library function that you utilize must be specified in code where they are used.
A. Line number 318 has the function `printf()`.

Q. How many arguments are we passing when we call printf() from main() in v2/, and what are their types?
A. The `printf()` from the `main()` in `v2/` passes four arguments which are char* type and three integer types for x, y and z.

Problem 6 (20 pts)

Q. The version in v3/ makes a further enhancement such that the two numbers to be subtracted are provided as input via standard input which, by default, is the keyboard of a Linux PC in the labs. From a logical perspective, **what is the role of & (i.e., ampersand) in**

scanf("%d %d",&x,&y)
A. The role of &(ampersand) in the scanf() function is giving the address of the storage where x is pointing. When the user types the integer value, the compiler stores the value at the address of the x which is &x.

Q. Using &(ampersand) in front of the variable will give us the address of the variable where the actual memory is located. Since the scanf() function should pass the address of the variable of x and y, we need the & in front of the variable name as an argument of scanf(). By passing the address of the variable, we can store the input into the address of the variables where the actual memory of the variables is located.

and why is

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

incorrect? Compile the code as is, and run it to verify that it works correctly. Then modify the code by removing the ampersand before x and compile main.c using gcc. What does gcc say after the modification?
A. If I remove the ampersand before the x and compile the code, the compiler gives me a warning and says:
`main.c: In function ‘main’:
main.c:14:11: warning: format ‘%d’ expects argument of type ‘int *’, but argument 2 has type ‘int’ [-Wformat=]

```
scanf("%d %d",x,&y);
        ~^`
```

This means that to get the input of `%d`, we need the address of the x which is int * type, not the `int` type.

Q. What happens when you run a.out? Explain your finding.
A. When I run a.out and type the first input, it gives me the following error: 9351 segmentation fault.
This means that the scanf() function in the code is not passing the address of the variable but just the variable. The reason why scanf() function should pass the address of the variable is that by passing the address of the variable, the compiler can point to the actual value of the variable by following the address of the variable. However, if scanf() function passes the variable itself(i.e.,

x or y), it will just point to the x which is just the storage of the actual value of x or y. Therefore, to find the actual value of x or y, we need to pass the address of the variable.

Problem 7 (20 pts)

Q. To output the content of the two int variables x and y, why is

printf("%d %d",x,y)

correct but

printf("%d %d",x,&y)

incorrect?

In the latter, what would we expect to be output?

A. The reason why the former is correct is that the printf() function passes the actual variable as the second argument and the third argument. Therefore, this is the reason why the latter one is not correct since there is the address of y instead of the actual value.
I expect the output to be the garbage value.

Q. Inside directory v2/ create a new file, main1.c, whose content is a copy of main.c. Modify main1.c so that an ampersand is placed in front of variable y when calling printf(). Compile and execute main1.c. How does the output differ from the original version main.c?

A. If I compile the main.c, it gives me the correct output. However, if I compile the main1.c, `&y` is always printing different values which are also called garbage values.

Problem 8 (20 pts)

Q. v4 contains a floating point (i.e., real number) variation of v3. Compile and test with real numbers to check that it works correctly.

A. Yes, it works correctly.
In v5, the code is made more modular by implementing the subtraction part of the app code as a separate function, subtwo(). Since subtwo() is essentially a one-liner, the separation does not buy much in terms of enhanced modularity. However, the principle is clear: if the calculations were more involved, putting the instructions in a separate function would make the design more modular and organized. subtwo() takes the values to be subtracted as its two arguments and returns the result to the calling function. Hence the caller is main() and the callee is subtwo().

Q. When passing the two arguments to subtwo(), why do we not use ampersands, that is, invoke subtwo(&x,&y)?

A. Since the subtwo() function is passing two float type variables as arguments, subtwo(&x,&y) will not work because &x and &y are pointer types. Therefore, when we compile with `subtwo(&x,&y)`, the compiler will give user the error and will not generate a.out file.
Another reason is that for scanf() function, we need to receive the input from the user and store this value to the address of the variable, so that is the reason why we have ampersand when we use scanf(). However, for subtwo() function, we pass two float arguments and just do the subtraction in the function. No more complicated work is needed for the function. That is why we don't have the ampersand in the subtwo() function.

Create a copy of main.c in v5/ and name the file main1.c. Modify main1.c so that main() calls subtwo() by placing ampersands in front of arguments x and y. Modify subtwo() so that it performs the subtraction calculation correctly. Compile main1.c and verify that it yields the desired result. Explain the logic behind your modification to subtwo() in main1.c.

A. I modified the code, and it runs correctly. The logic is if I put an ampersand in front of the arguments x and y on the line, we call the subtwo function, this means that I will pass the address of x and y where the values are located. Since subtwo() function is passing the address of the x and y, we need to change the code of where the subtwo() function was made, line number 30. Since we get the address of the variable, we need to put the asterisk(*) in front of the variable name so that we can point the address of the variable. By doing this way, I can point the value of the variable, not the address. Since I changed the arguments on line 30, I also need to change the variable at line 36. But I don't have to change the variable c since it is already initialized as a float type at line 32. Moreover, I need to change the variables in arguments on line 7 since the subtwo() function is passing the address of the variables, not the actual value.

Problemi 9 (20 pts)

Q. Describe what the modification performed in v6 is.

A. In v5, all codes and every function were in one file called main.c. However, in v6, there are two separate .c files. One is the main.c which contains the main() function. Another one is the subtwo.c which contains the subtwo function.

Q. How should the code of v6 be compiled?

A. We can link those two separate c files by using linking. If I type `gcc main.c subtwo.c`. those two separate c files will be compiled together and will make one a.out which will be executed successfully.

Q. Create a subdirectory, v6sub, under v6/ and move the file subtwo.c from v6/ to v6/v6sub/. From within folder v6/ how do you need to run gcc so that it compiles the code correctly by finding all the files it needs?

A. Since I need to find all the files which has an extension of .c, I can use this following command.

`gcc *c v6sub/*.c`. Asterisk(*) means all the files. If I do `*.c`, then this means all the files ends with `.c`. Therefore, by typing the command, we can find all the files the compiler needs and compile it correctly.

Q. What final modification is carried out in v7?

A. Header file which is myheader.h is created. In v6/, `float subtwo(float, float);` this line of code was in the main.c. However, in v7, the line of code was moved in to myheader.h.

Q. Create a subfolder v7sub/ under v7/, and move the files subtwo.c and myheader.h into v7sub/. What happens if you try to compile the app as in Problem 8?

A. Compiler gives me this following error. `fatal error: myheader.h: No such file or directory`. How I compiled in Problem 8 is just typing `gcc main.c`. If I do this command in v7/, then it should give me the error since main.c is including `myheader.h` file at the beginning of the main.c which is in the different directory from the directory of main.c.

Q. What modification must be made to main.c so that compilation succeeds? Make the modification and verify that your code works correctly.

A. In main.c, on the line where include “myheader.h”, since the `myheader.h` is in the subfolder of `v7/` where the `main.c` is located, I changed `#include " myheader.h"` to `#include "v7sub/myheader.h"`, so that the preprocessor can find where the `myheader.h` is located.

Bonus problem (20 pts)

Q. Code a function, float mintwo(float a, float b), that returns the minimum of its two float arguments to the caller. Place the function in its own file mintwo.c under a new directory v8/ in lab1/. Code a test program main() in main.c in v8/ that calls mintwo() with values 333.3 and 11.2. Compile and test that your code works correctly.

The Bonus Problem is entirely optional. Bonus points count toward reaching 45% of the course grade contributed by lab assignments.