Problem 1

1. **What is the size of the output, i.e., file a.out, generated by gcc?**
   8.0K

2. **What is the full pathname of a.out?**
   /homes/khan274/cs240/lab1/v1/a.out

3. **What happens when you try to inspect the content of the file a.out by using the app cat, and why?**
   The screen displays a combination of question mark symbols and characters that don't make sense. This is because a.out only contains bytes, "computer language", and cat is not good at translating it back into something understandable.

4. **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?**
   The main.c file is written by humans as code. GCC can take this code and translate it into an executable file. This file contains bytes that the computer can understand and use to perform the tasks specified in the human-written code.

5. **In what sense is the C program contained in the file main.c not very useful?**
   main.c is not very useful to us because it never tells us the result of the calculation it performs. It's like if you had a calculator that never displayed any results--you would have no use for it.

Problem 2

1. **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/.**

   I ran chmod 700 * in my lab1 folder. Afterwards it said the permissions for my directories v1 - v7 were all "drwx--S--- 2". I believe what this means is that only I can read, write, and execute files in these directories, due to the "d" specifying directory, and the first three characters afterwards pertaining to me being "rwx", meaning I can read, write, and execute. The next three characters pertain to a group, and the r and w are absent, meaning they cannot read or write. I believe the S means they can have some sort of permissions with regards to executing. The next three characters are all dashes, which I believe means that any other user cannot read, write, or execute the contents of these folders in any way.

Problem 3

1. **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. 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.

The shell I am running is bash. We don't expect much to happen materially, since the computer will not output anything when the executable will run. Behind the scenes, it will do some math and store the result of the 7 - 3 in the memory location of z.

Problem 4

1. **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. Given that main() is where supposedly execution begins, why is it that main() itself is declared to return a value of type int?**

   The main function returns an int to let the operating system know whether the program exited successfully. If it returns a 0, it was successful, otherwise it was not.

2. **Note that in v1/main.c the last statement of main() is "return 0". 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.**

   I get main.c:6:1: warning: return type defaults to 'int' [-Wimplicit-int] and an arrow pointing to right before main(). It did not compile.

3. **What happens if you modify the spelling of main() to Main() and compile? Did compilation succeed? Explain.**

   I got a long series of characters that look like a path name, and then "undefined reference to 'main' collect2: error: ld returned 1 exit status". It did not compile.

Problem 5

1. **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 library 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. Find where the file stdio.h is located, i.e., its full pathname. How large is the file?**

   Pathname: /usr/include/stdio.h

Size: 29665

2. **What are its protection bits, and what do they convey?**
   They are -rw-r--r--, which means that the owner can read and write in the file, and everyone else can only read them.

3. **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. How many arguments are we passing when we call printf() from main() in v2/, and what are their types?**

   It is on line 318 in the stdio.h file.
   We pass 3 arguments of type int.

Problem 6

1. **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)

   and why is

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

   incorrect?

   The role of the & is that it denotes the address that x is assigned to, rather than just x itself. The number input from stdin has to be assigned to an address in memory, which &x denotes, whereas x alone is just the name assigned to that piece of memory. You can't store an int directly into it because of that.

2. **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? What happens when you run a.out? Explain your finding.**

   Gcc says: warning: format '%d' expects argument of type 'int *', but argument 2 has type 'int' [-Wformat=]
   scanf("%d %d",x,&y);

When I run a.out the program gets stuck on scanf, doesn't print anything, and doesn't exit. I believe this is because it doesn't know which address to store the input in, since without the & there is no address specified.

Problem 7

1. **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?

   When you include the & before y, it points to the address that the value you are trying to print is stored in, not the value itself.

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

   The program would print the value of x, followed by the address of y rather than the value.

3. **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?**

   To start with, gcc prints a warning:
   main1.c:19:27: warning: format '%d' expects argument of type 'int', but argument 3 has type 'int *' [-Wformat=]
   printf("result of %d - %d equals %d\n", x, &y, z);

   When I run a.out after that, it says:
   result of 7 - -1661903860 equals 4

   This is compared to main.c output:
   Result of 7 - 3 equals 4

   These results verify my answer to part 2, where I noted that it would print the address rather than the value of y.

Problem 8

1. **v4 contains a floating point (i.e., real number) variation of v3. Compile and test with real numbers to check that 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 doesn't 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 substracted as its two arguments and returns the result to the calling function. Hence the caller is main() and the callee is subtwo(). When passing the two arguments to subtwo(), why do we not use ampersands, that is, invoke subtwo(&x,&y)?**

   Subtwo is meant to subtract the values of x and y, not the addresses of x and y. It's similar to how when passing &y into printf printed the address rather than the value--you would not be accessing the number that you intend to.

2. **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.**

   The changes I made to make it work was changing the function declaration to:

   float subtwo(float *, float *)

   Within the subtwo function, I changed c = a - b to c = *a - *b, because when you read in subtwo(&x,&y), it will pass the addresses of x and y. I wanted to get the values at those addresses to subtract, so I used the * to "dereference" a and b and then subtracted. I had to change the function declaration to match the new variable types getting passed.

Problem 9

1. **Describe what the modification performed in v6 is. How should the code of v6 be compiled?**

   In v6, main.c and subtwo.c are two different files. main.c calls subtwo but the code for subtwo that was previously below the main function is now in its own file. You can compile it with:

   gcc main.c subtwo.c

2. **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?**

   I used:
   gcc main.c v6sub/subtwo.c

3. **What final modification is carried out in v7?**

   v7 contains a file called myheader.h.

4. **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?**

   When I try to use gcc main.c v7sub/subtwo.c I get an error saying:
   main.c:7:10: fatal error: myheader.h: No such file or directory
   #include "myheader.h"

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

   I changed #include “myheader.h” to #include “v7sub/myheader.h” and compilation worked with gcc main.c v7sub/subtwo.c.

Bonus Question
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.