Problem 1 (30 pts)

Compile the code in directory v1 by running
% gcc main.c
What is the nature of the output a.out generated by gcc?
The result is unreadable when opened in vim, and cannot be run.
What are the three steps performed by gcc that results in a.out?
GCC, which is a compiler, takes various source code documents given to it, converts them to object code, sends them to an associated linker, which converts the object code into one executable
What does each of the steps accomplish?
When source code is compiled into an executable, it must be translated/interpreted by a compiler, given to a linker, and put together into a singular coherent file.
What are two methods to change the name of the output from a.out to, say, main.exe?
Using the -o tag followed by the desired name:
gcc main.c -o main.exe
You could also use the Unix *mv* command to rename the file (after it has been created)
mv a.out main.exe
What are all the reserved keywords per C language syntax in main.c?
int, main
What kind of output is produced by gcc when main.c is compiled with the option -S:
A file named main.s is produced which seems to be some sort of translation of the code in main.c , possibly Assembly.
What kind of output is produced when gcc option -c is used?
A file named main.o is produced. (-c is used to tell the compiler to only compile, and *not* to link)
What happens when you try to run the output generated by gcc with the -c option?
The output generated by running *gcc -c main.c*, main.o, is unreadable in vim.

Problem 2 (30 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(). printf() outputs to standard output which is, by default, the terminal (i.e., display or screen) associated with a PC.
Where is the header file stdio.h located on our Linux lab machines in LWSN B148?
Stdio.h is located with the C library files located on the lab machines.
What component of gcc handles processing of #include <stdio.h> to find and insert the content of stdio.h into main.c?
The C preprocessor, which is automatically called by the compiler.
Perform gcc with the -E and -P options and save the output (sent to the terminal) into a file puremain.c using the ">" redirection operator of the shell you are using. Inspect puremain.c and explain how it is different from main.c.
It contains a massive 300+ line header, which I'm assuming contains detailed information and attributes not explicitly visible in the "simplified" main.c file.

What part of "result of %d - %d is %d" is literal (just characters to be printed as is) and what part is reserved, i.e., part of formatting control of C that determines how content of a variable is output?
%d is the reserved character used in this line, which refers to additional variables that would be attached to the end of the line, separated by commas.
Where is the actual compiled object code of printf() located on our Linux machines? Make use of the ldd utility.
/lib64/libc.so.6
Based on the file extension of the library where printf() is located, is the object code of printf() statically or dynamically linked with main.o?
They are dynamically linked.
Why is one method preferred over the other?
Dynamic linkage uses less memory
Compile and run main.c, and check that it works correctly.

Problem 3 (30 pts)

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

scanf("%d %d",&x,&y)
and why is
scanf("%d %d",x,y)

Incorrect?
the ampersand serves as the primary reference operator in C
Compile the code as is, and run it to verify that it works correctly. Modify the code by removing the two ampersands and compile main.c using gcc. What does gcc say after the modification?
It says it expects an argument of type int, which means that it is not properly reaching the values inputted by the user.
Note that as long as it's a warning, gcc produces a new executable. What happens when you run a.out? What might have happened to cause the run-time error you observe? Why are ampersands not used in printf()?
Printf() is referencing the static integers already assigned by the scanf statement, which requires the ampersands to properly assign the integers provided by the user to memory

Problem 4 (50 pts)

v4 contains a floating point (i.e., real number) variation of v3. Compile and test with real numbers (say, 7.3 and 4.1) 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, mycalc(). Since mycalc() is essentially a one-liner, the separation doesn't buy much in terms of clean design. However, the principle is clear: if the calculations were more involved, putting the instructions in a separate function makes the design more modular. mycalc() takes the values to be subtracted as its two arguments and returns the subtraction to the

calling function. Hence the caller is main() and the callee is mycalc(). When passing the two arguments to mycalc(), why do we not use ampersands, that is, mycalc(&x,&y)?
Similarly to the printf() that we spoke about earlier, the values referenced by mycalc() have already been assigned, and thus & is not required.
Suppose instead of communicating to the caller the subtraction result using a function return, we want the callee to directly update the variable z in main() with the computed result. One way to do it is to change

```
z = mycalc(x,y);
```
to
```
mycalc(x,y,&z);
```

Noting the meaning of &, why should adding &z as the third argument of mycalc() enable the callee to update the value of variable z which belongs to main()? If this method of communicating between caller and callee is employed, C requires that the code of the callee be updated as follows:

```
void mycalc(float a, float b, float *c)
{
  *c = a - b;
}
```

The * symbol preceding argument c in the callee specifies that c contains an address (not value). Thus we are telling the callee mycalc() where in RAM (i.e., memory) z is located, i.e., its address. Since the callee has this information, it can now, if it wants to, go to the address and update the content at that address. This is done by prepending c with * and performing the subtraction *c = a - b, instead of c = a - b. Make the modifications and compile the updated main.c and mycalc.c. What does gcc say?
An error is given, specifying that too many arguments were passed.
What additional modification do we need to make to have a correct code? Figure it out on your own and if you get stuck the TAs will help. Compile the updated code and test that it runs correctly. Note that type of mycalc() has been changed from float to void to explicitly specify that it does not return a value.
float mycalc(float, float);
should be changed to
float mycalc(float, float, float *);

Problem 5 (10 pts)

Describe what the modification performed in v6 is.
Mycalc is now in a separate file
How should the code of v6 be compiled?
Both files have to be compiled in tandem, i.e: *gcc main.c mycalc.c*
What final modification is carried out in v7.
Myheader.c is added.
Explain what the difference between

#include "mydecl.h"
and
#include <mydecl.h>

is. What happens if you replace by angle brackets of stdio.h by double quotes? What happens if you replace the double quotes of mydecl.h by angle brackets?
Brackets refers to a file imported from the C Library, while the double quotes will refer to a file in the same directory. If either symbol is changed, it will cause the program to not function in the way originally intended.

Bonus problem (20 pts)

Modify the version in v7 so that instead of subtracting two numbers of type float, the app calculates the sine of the first number, the cosine of the second number, then adds the two results which are stored in z and printed to the display (i.e., standard output).

The function definitions of main() and mycalc() should stay the same, and the same goes for reading the input and writing the output. The sine and cosine functions are available as library functions in <math.h> but need to be compiled with a specific option when running gcc. Consult the man page on our Linux machines to figure out the correct usage of library functions (use the float versions) and their compilation. Create a subdirectory v8 where your modified code (all *.c and *.h files) are located that will compile and run correctly when the TAs test them.

Bonus problem attempted
Executable was generated by running

```
gcc -lm main.c mycalc.c
```