Raman Kahlon

CS 240 Lab 1

Problem 1:

The nature of the output a.out generated by gcc is executable code in binary that runs on the CPU. The three steps performed by gcc that results in a.out are the C preprocessor, C compiler, and the linker. The C preprocessor looks for the # symbol, such as in the header declaration #include <stdio.h>, and sends the input to the file. Then, the C compiler translates the preprocessed code into assembly instructions in an intermediate language. Finally, the linker assembles the pieces of object code in order so that the functions in some pieces of code can call functions in other pieces. The linker also fills in the missing pieces of the program to create a functional program.

If you omit the type declaration of main(), gcc will give a warning: "return type defaults to 'int'" for main. The gcc will still generate an executable. If you declare main()'s type as void, gcc will not give an error.

Problem 2:

The header file stdio.h is located on the Microsoft Visual Studio application on our Linux lab machines in LWSN B148.

The literal part of the String "result of %d + %d is %d" is "result of", "+", and "is". The symbol "%d" is reserved, as the three times it is utilized in the String represents each of the three variables (of type int) included as parameters of the printf() function. If you run gcc with option -c, it will give a warning: "main: 'linker' input file unused because linking not done". You can execute the output generated by gcc with option -o, but you can't execute the output generated by gcc with option -c because the linking stage is not done.

Problem 3:

The ampersand ('&' symbol) is used to locate the address of the variable. The function scanf("%d %d",x,y) is incorrect because x and y are simply symbols, or placeholders, for the addresses in the RAM. With &x and &y, you can locate the address of each variable. If you remove the two ampersands, gcc gives the warning: "format '%d' expects argument of type 'int *', but the argument has type 'int'". This warning appears twice, once for each variable, x and y. This means that x and y should be pointers when used as parameters in scanf(), not simply variables.

When you run a.out, nothing is printed because the CPU is unable to locate the addresses of the variables x and y. Since the addresses can't be located, x and y have no bit patterns to hold their values and a.out fails to properly execute.

Problem 4:

The function printf() in v3/main.c doesn't use ampersand before the variables x, y, and z because printf() only needs the values of the variables, not the addresses of the variables, in order to print them out. The '&' functions as a reference operator, representing the variable's memory location. If you add '&' before the three local variables, gcc will give the warning: "format '%d' expects argument of type 'int', but the argument has type 'int *'". This warning appears three times, once for each variable x, y, and z. This indicates that x, y, and z should be represented as variable parameters in printf(), not pointers. If you try to compile the code this way, gcc will still give three warnings and will not generate an executable.

Problem 5:

When passing the two arguments, x and y, to myadd(), we don't use ampersands because the arguments are not supposed to be pointers. In C, arguments are passed to functions by value, so the function myadd() can't affect the arguments x and y in the caller function main(). Thus, calling myadd() without ampersands would simply add copies of x and y and place them into the variable z.

Adding &z as the third argument of myadd() enables the callee to directly update the value of variable z which belongs to main() because &z is used as a pointer to the address of the variable z. By changing the addition in myadd() from c = a + b to *c = a + b, gcc gives four errors, two regarding the incompatible types of parameters between argument 3 of the myadd() call and the myadd() declaration (float and float *) and the other two regarding the return type of the method myadd(). This is because the return type of myadd() in the method declaration is stated as void, while the actual method in the program has a return type of int. In order to have correct code, we must change the declaration of myadd() to the return type of int and change the third parameter from 'float' to 'float *'. Since myadd() does not return a value, we can assign the return type of void to myadd().

Problem 6:

The modification performed in v6 is that the function myadd() is put into a separate file, myadd.c. The code of v6 should be compiled by using the command: gcc myadd.c main.c, since main.c is the file that calls the function myadd() in myadd.c. The final modification in v7 is that the declaration of the myadd() function is placed into its own separate file, myheader.h. The headers #include “mydecl.h” and #include <mydecl.h> are different because “mydecl.h” is used for header files of the user’s own program, searching in the current file’s path. This means that the compiler will look for a file in the local directory. Meanwhile, <mydecl.h> is used for system header files, which searches in global include files. If you replace the angle brackets of stdio.h with double quotes, the preprocessor will search for stdio.h in the same directory as the file containing the directive. If you replace the double quotes of stdio.h with angle brackets, the preprocessor will search in an implementation dependent manner, normally in search directories pre-designated by the compiler/IDE.

Bonus Problem:

In order to pass all arguments as pointers, I first declared the function myadd() as void myadd(float *, float *, float *) to indicate that myadd() needs to be passed three pointers. I then changed the header of the myadd() function itself to void myadd(float *a, float *b, float *c) for the three pointers. In the body of the function, I changed *c = a + b into *c = *a + *b because a and b now represent two different pointers to the two numbers of input. Just like in v5 before, c remains a pointer to an address that contains the sum of the input numbers a and b.