Exercises-Functions

Part One
In this part you will practice simple function declarations and what happens in the presence or absence of a function prototype. The example programs are in the functions directory beneath the class public data area on hills.

1. Examine the program noproto.c  Predict what it outputs. Run the executable noproto. Did it perform as expected?
2. Repeat this procedure for the program proto.c
3. Make a copy of proto.c and place the function main() before the function output() in proto.c  Predict if this will cause a problem and then try it.
4. As you discovered, the compiler complains once you rearrange the functions. Even though the program compiles, now you get an incorrect answer! How can you fix it, leaving the order of the functions as they are? Try your fix to see if it works.
5. Copy the files fprintf1.c and fprintf2.c to your home directory.
6. Examine fprintf1.c. You should be able to immediately see the problem with this program. Compile it and run it. Note that no compile-time warning is given, even though the result is totally incorrect.
7. Examine fprintf2.c. Again, you should be able to see the problem. Compile this program, and run it if you like. Note that the argument types of the first two arguments to fprintf() are checked, and the compiler gave you a warning that they were incorrect.
8. Examine the stdio.h include file and search for the prototype of fprintf. (use /fprintf in vi to search for the function prototype). This prototype is a type that we have not covered. Note the ellipses at the end of the prototype. What do you think this means?

Part Two
After doing part one and reviewing your notes on functions, explain the following situations:
1. printf has a single conversion specifier for float and double, %f. Why is this sufficient?
2. if your system supports a long double and you want to print it, you must specify %Lf. Why?
3. you can output data of type char, short, int and long using %d. Why?
4. What is the purpose of the %c conversion specifier, given the previous question?
5. You are calling a function with the following interface: float foo(double d). Which of the following declarations of foo must have been seen prior to calling it in order for the function to perform correctly (i.e., get the correct arguments and return the correct result) when it is called like this: float f=4.0;  f=foo(f);
   • the function will perform correctly even if no declaration of foo() has been seen.
   • only the return type must have been specified in a declaration; e.g., float foo(); is sufficient.
   • the entire prototype must have been specified; e.g., float foo(double);
6. The function moo has the following prototype: double moo(void). The prototype is ‘in-scope’ (this means it has been seen and is effective at the current position in the compilation). Which of the following calls to moo are reasonable (i.e., will return values correctly)?
   • int i = moo();
   • moo();
   • double d=moo();
7. Having learned the inside information on what is necessary for a function call to succeed, why is it always a good idea to only call functions when complete prototypes have been declared for them?
Answers and Hints

Part One

1. The function output() in noprotot.c does not have a function prototype. Thus, when it is called, the default argument promotions are used for the arguments and the actual types of the function’s arguments are treated as unknown. This is true even though the function declaration precedes the function call, since the function declaration is not in prototype form. According to the default argument promotions, an int is just passed as an int and a double is just passed as a double. The attempt to interpret the [passed] int as a double causes a problem that is not caught by the compiler.

2. Here output() uses prototype form. Thus, when it is called with an int, the int is converted to the type the function declared for the first argument (double), and everything works out.

3. <no notes>

4. After you reverse the order of the functions, the code for the call is generated prior to the declaration of the argument type by the prototype form of the function definition. When the call is seen, an implicit declaration of output() is created, without a prototype. When the prototyped definition is encountered, the compiler complains that the argument type has been redeclared.

5. You can fix the program by inserting a function declaration in prototype form prior to main(). It looks like this

    void output (double);  OR
    void output (double xxxx);  /* xxxx is just a dummy identifier */

    Try inserting this prototype before main() and see that the program now works correctly.

6. <no notes>

7. <no notes>

8. The ellipses found in the prototype for fprintf in /usr/include/stdio.h are used to denote a variable argument list. In a variable argument list, there are some number of declared arguments that must be present, followed by zero or more untyped arguments. The compiler, of course, can only check the types of (and convert if necessary) the arguments that have them. The arguments that occur in the variable argument area are passed using the default argument promotions.

Part One

1. The arguments being converted in a call to printf (i.e., all arguments except the format string) are not typed in the prototype. This means that the default argument conversions apply: all floating-point values of size shorter than double are converted to double. Thus, %f only outputs double. If you call printf with a float, the float gets converted to double before the call occurs.

2. long double is another story. Since it results in a special data type being passed, it must have its own conversion specifier.

3. Again, all integral types whose size is shorter than an int are converted (extended) to the size of int before the call. %d only prints ints. HOWEVER, if you have an unsigned value whose size is the same as an int, you should use %u

4. The %c conversion specifier basically says: take the integer argument passed and translate its least-significant 8 bits to the corresponding character value. Try using printf with the value 0x4241 (0x41 is the hex value for ascii 'A')

5. The return type must be declared. If it is not, the return type is assumed to be int, and the value will be converted before storing in f. Since the type of foo’s single argument is the same as the type resulting from the default argument promotion of a float, the declaration of the argument type in the prototype was superfluous.

6. All of the example calls will produce the correct result, since moo was called when its prototype was in-scope.
7. You should always make sure your functions have complete prototypes in-scope. This allows the compiler to insert appropriate conversions and to give you an error if a conversion cannot be performed. It also generates better information for the debugger in the off-chance you will have to use it to debug your program.