11.1 Describe some problems or inconveniences you found when programming in lower-level languages.

11.2 How do higher-level languages help reduce the tedium of programming in lower-level languages?

11.3 What are some disadvantages to programming in a higher-level language?

11.4 Compare and contrast the execution process of an interpreter versus the execution process of a compiled binary. What implication does interpretation have on performance?

11.5 A language is portable if its code can run on different computer systems, say with different ISAs. What makes interpreted languages more portable than compiled languages?

11.6 The UNIX command line shell is an interpreter. Why can’t it be a compiler?

11.7 Is the LC-3 simulator a compiler or an interpreter?

11.8 Another advantage of compilation over interpretation is that a compiler can optimize code more thoroughly. Since a compiler can examine the entire program when generating machine code, it can reduce the amount of computation by analyzing what the program is trying to do.

The following algorithm performs some very straightforward arithmetic based on values typed at the keyboard. It outputs a single result.

1. Get W from the keyboard
2. X ← W + W
3. Y ← X + X
4. Z ← Y + Y
5. Print Z to the screen

a. An interpreter would execute the program statement by statement. In total, five statements would execute. At least how many arithmetic operations would the interpreter perform on behalf of this program? State what the operations would be.

b. A compiler would analyze the entire program before generating machine code, and possibly optimize the code. If the underlying ISA were capable of all arithmetic operations (i.e., addition, subtraction, multiplication, division), at least how many operations would be needed to carry out this program? State what the operations would be.

11.9 For this question refer to Figure 11.2.

a. Describe the input to the C preprocessor.
b. Describe the input to the C compiler.
c. Describe the input to the linker.
What happens if we changed the second-to-last line of the program in Figure 11.3 from `printf("%d\n", counter);` to:

a. `printf("%c\n", counter + 'A');`

b. `printf("%d
%d\n", counter, startPoint + counter);`

c. `printf("%x\n", counter);`

The function `scanf` reads in a character from the keyboard and the function `printf` prints it out. What do the following two statements accomplish?

```c
scanf("%c", &nextChar);
printf("%d\n", nextChar);
```

The following lines of C code appear in a program. What will be the output of each `printf` statement?

```c
#define LETTER 'i'
#define ZERO  0
#define NUMBER 123

printf("%c", 'a');

printf("%x", 12388);  
printf("%d.%c%hd\n", NUMBER, LETTER, ZERO);
```

Describe a program (at this point we do not expect you to be able to write working C code) that reads a decimal number from the keyboard and prints out its hexadecimal equivalent.