13.6 Summary

We conclude this chapter by summarizing the key concepts we've covered. The basic objective of this chapter was to enlarge our set of problem-solving primitives by exploring the various control structures supported by the C programming language.

- **Decision Construct in C.** We covered two basic C decision statements: `if` and `if-else`. Both of these statements *conditionally* execute a statement depending on whether a specified expression is true or false.

- **Iteration Constructs in C.** C provides three iteration statements: `while`, `for`, and `do-while`. All of these statements execute a statement possibly multiple times until a specified expression becomes false. The `while` and `do-while` statements are particularly well-suited for expressing sentinel-controlled loops. The `for` statement works well for expressing counter-controlled loops.

- **Problem Solving Using Control Structures.** To our arsenal of primitives for problem solving (which already includes the three basic C types, variables, operators, and I/O using `printf` and `scanf`), we added control constructs. We practiced some problem-solving examples that required application of these control constructs.

---

**Exercises**

13.1 Recreate the LC-3 compiler's symbol table when it compiles the calculator program listed in Figure 13.24.

13.2  

a. What does the following code look like after it is processed by the preprocessor?

```c
#define VERO -2

if (VERO)  
    printf("True!");
else  
    printf("False!");
```

b. What is the output produced when this code is run?

c. If we modified the code to the following, does the code behave differently? If so, how?

```c
#define VERO -2

if (VERO)  
    printf("True!");
else if (!VERO)  
    printf("False!");
```
13.3 An if-else statement can be used in place of the C conditional operator (see Section 12.6.3). Rewrite the following statement using an if-else rather than the conditional operator.

\[ x = a ? b : c; \]

13.4 Describe the behavior of the following statements for the case when \( x \) equals 0 and when \( x \) equals 1.

a. if (\( x = 0 \))
   
   \[ \text{printf("x equals 0\n");} \]
   
   else
   
   \[ \text{printf("x does not equal 0\n");} \]

b. if (\( x == 0 \))
   
   \[ \text{printf("x equals 0\n");} \]
   
   else
   
   \[ \text{printf("x does not equal 0\n");} \]

c. if (\( x == 0 \))
   
   \[ \text{printf("A\n");} \]
   
   else if (\( x != 1 \))
   
   \[ \text{printf("B\n");} \]
   
   else if (\( x < 1 \))
   
   \[ \text{printf("C\n");} \]
   
   else if (\( x \))
   
   \[ \text{printf("D\n");} \]

d. int \( x \);

   int \( y \);

   switch (\( x \)) {
   
   case 0:
   
   \[ \text{y = 3; }
   
   case 1:
   
   \[ \text{y = 4; }
   
   \text{break; }
   
   default:
   
   \[ \text{y = 5; }
   
   \text{break; }
   
   }

e. What happens if \( x \) is not equal to 0 or 1 for part 4?

13.5 Provide the LC-3 code generated by our LC-3 C compiler when it compiles the switch statement in part 4 of Exercise 13.4.

13.6 Figure 13.12 contains a C program with a nested for loop.

a. Mathematically state the series that this program calculates.

b. Write a program to calculate the following function:

\[ f(n) = f(n - 1) + f(n - 2) \]

with the following initial conditions,

\[ f(0) = 1, \quad f(1) = 1 \]
13.7 Can the following if-else statement be converted into a switch? If yes, convert it. If no, why not?
if (x == 0)
    y = 3;
else if (x == 1)
    y = 4;
else if (x == 2)
    y = 5;
else if (x == y)
    y = 6;
else
    y = 7;

13.8 At least how many times will the statement called loopBody execute the following constructs?
a. while (condition)
   loopBody;
b. do
   loopBody;
   while (condition);
c. for (init; condition; reinit)
   loopBody;
d. while (condition1)
   for (init; condition2; reinit)
   loopBody;
e. do
do
   loopBody;
   while (condition1);
   while (condition2);

13.9 What is the output of each of the following code segments?
a. a = 2;
   while (a > 0) {
       a--;
   }
   printf("%d", a);
b. a = 2;
   do {
       a--;
   } while (a > 0)
   printf("%d", a);
c. b = 0;
   for (a = 3; a < 10; a += 2)
       b = b + 1;
   printf("%d %d", a, b);
13.10 Convert the program in Figure 13.4 into one that uses a `switch` statement instead of `if-else`.

13.11 Modify the e-mail address validation program in Figure 13.23 so that it requires that at least one alphabetic character appears prior to the at sign, one appears between the at sign and the period, and one appears after the period in order for an e-mail address to be valid.

13.12 For the following questions, `x` is an integer with the value 4.

   a. What output is generated by the following code segment?

   ```
   if (7 > x > 2)
       printf("True.");
   else
       printf("False.");
   ```

   b. Does the following code cause an infinite loop?

   ```
   while (x > 0)
       x++;
   ```

   c. What is the value of `x` after the following code has executed?

   ```
   for (x = 4; x < 4; x--) {
       if (x < 2)
           break;
       else if (x == 2)
           continue;
       x = -1;
   }
   ```

13.13 Change this program so that it uses a `do-while` loop instead of a `for` loop.

   ```
   int main()
   {
       int i;
       int sum;

       for (i = 0; i <= 100; i++) {
           if (i % 4 == 0)
               sum = sum + 2;
           else if (i % 4 == 1)
               sum = sum - 6;
           else if (i % 4 == 2)
               sum = sum + 3;
           else if (i % 4 == 3)
               sum = sum / 2;
       }
       printf("%d\n", sum);
   }
   ```
chapter 13  Control Structures

13.14 Write a C program that accepts as input a single integer \( k \), then writes a pattern consisting of a single 1 on the first line, two 2s on the second line, three 3s on the third line, and so forth, until it writes \( k \) occurrences of 5 on the last line.

For example, if the input is 5, the output should be the following:

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

13.15 a. Convert the following while loop into a for loop.

```c
while (condition)
    loopBody;
```

b. Convert the following for loop into a while loop.

```c
for (init; condition; reinit)
    loopBody;
```

13.16 What is the output of the following code?

```c
int x = 0;
int s = 0;
int w = 12;
int sum = 0;

for (x = 1; x <= w; x++)
    for (s = x; s <= w; s++)
        sum = sum + s;

printf("sum=%d\n", sum);
```

13.17 The following code performs something quite specific. Describe its output.

```c
int i;

scanf("%d", &i);
for (j = 0; j < 16; j++)
    if (i & (1 << j)) {
        count++;
    }

printf("%d\n", count);
```
13.18 Provide the output of each of the following code segments.

a. int x = 20;
   int y = 10;
   while ((x > 10) && (y & 15)) {
      y = y + 1;
      x = x - 1;
      printf("**");
   }

b. int x;
   for (x = 10; x > 1; x = x - 1)
      printf("**");

c. int x;
   for (x = 0; x < 10; x = x + 1) {
      if (x % 2)
         printf("**");
   }

d. int x = 0;
   int i;
   while (x > 10) {
      for (i = 0; i < x; i = i + 1)
         printf("**");
      x = x + 1;
   }