CHAPTER 2

Control Statements

This chapter presents recipes that exploit the power of control statements to solve problems. C is rich in control statements. Control statements in C can be broadly classified into three categories, as follows:

- Selection statements
- Iteration statements
- Jump statements

Selection Statements

A selection statement is used to choose one of the several flows of computer control. There are two selection statements: `if-else` and `switch`.

Iteration Statements

An iteration statement is used to execute a group of statements repeatedly, a finite number of times. There are three iteration statements: `while`, `do-while`, and `for`.

Jump Statements

There are four jump statements: `break`, `continue`, `goto`, and `return`. Normally computer control flows linearly from the preceding statement to the next statement in the source code. You use a jump statement when you need to bypass this linear flow and have the computer control jump from one statement to another statement, not necessarily the successive one.

The `goto` statement is used to jump to another statement within the same function. The `continue` statement is used only in iteration statements. The `break` statement is used only in iteration or `switch` statements. The `return` statement is used in functions.

S. Chavan, *C Recipes*, DOI 10.1007/978-1-4842-2967-5_2

2-1. Sum 1 to N Numbers

Problem

You want to develop a program that computes the sum of 1 to N numbers in an interactive manner.

Solution

Write a C program that computes the sum of 1 to N numbers with the following specifications:

- The program uses the `for` loop to perform the summation of 1 to N numbers. Nothing is sacred about the `for` loop; you can also use the `while` loop or the `do-while` loop, but in these type of programs the `for` loop is most preferred.
- The program asks the user to enter the number N (0 < N < 30000). If the user enters the number N outside of this range, then the program asks the user to reenter the number.
- When the computed sum is displayed on the screen, the program asks the user whether he or she wants to compute another sum or quit.

The Code

The following is the code of the C program written with these specifications. Type the following C program in a text editor and save it in the folder `C:\Code` with the file name `sum.c`:

```
/* This program computes the sum of 1 through N numbers using for statement in an */
/* interactive manner. */
                                                                    /* BL */
#include <stdio.h>                                                  /* L1 */
                                                                    /* BL */
main()                                                              /* L2 */
{                                                                   /* L3 */
 int intN, intCounter, flag;                                        /* L4 */
 unsigned long int ulngSum;                                         /* L5 */
 char ch;                                                           /* L6 */
                                                                    /* BL */
 do {                       /* outer do-while loop begins */       /* L7 */
                                                                    /* BL */
  do {                      /* inner do-while loop begins */       /* L8 */
   flag = 0;                                                        /* L9 */
   printf("Enter a number (0 < N < 30000): ");                      /* L10 */
   scanf("%d", &intN);                                              /* L11 */
   if ((intN <=0) || (intN > 30000))                                /* L12 */
     flag = 1;                                                      /* L13 */
  } while (flag);           /* inner do-while loop ends */         /* L14 */
                                                                    /* BL  */
```

```
  ulngSum = 0;                                                    /* L15 */
                                                                  /* BL */
  for (intCounter = 1; intCounter <= intN; intCounter++) {        /* L16 */
   ulngSum = ulngSum + intCounter;                                /* L17 */
  }                                                               /* L18 */
                                                                  /* BL */
  printf("Required sum is: %lu\n", ulngSum);                      /* L19 */
  printf("Do you want to continue? (Y/N) : ");                    /* L20 */
    scanf(" %c", &ch);                                            /* L21 */
 } while ((ch == 'y') || (ch == 'Y'));  /* outer do-while loop ends */ /* L22 */
                                                                  /* BL  */
 printf("Thank you.\n");                                          /* L23 */
 return(0);                                                       /* L24 */
}                                                                 /* L25 */
```

Compile and execute this program. A "run" of this program is given here:

```
Enter a number (0 < N < 30000): 10000   ↵
Required sum is: 50005000
Do you want to continue? (Y/N) : y   ↵
Enter a number (0 < N < 30000): 31000   ↵
Enter a number (0 < N < 30000): 25000   ↵
Required sum is: 312512500
Do you want to continue? (Y/N) : n   ↵
Thank you.
```

How It Works

The `for` loop contained in LOCs 16 to 18 performs the summation of 1 to N numbers. `do-while` loops with two-level nesting are used in this program. The inner `do-while` loop keeps the user inside the loop as long as the user fails to enter the number N in the specified range. The outer `do-while` loop keeps the user inside the loop as long as the user wants to perform the summation again. The inner `do-while` loop adds robustness to this program. Besides the `for` loop, you can also use `while` or `do-while` loops to perform the summation. To use a `while` loop to perform summation, replace LOCs 16 to 18 with the following LOCs:

```
intCounter = 0;
while (intCounter < intN) {
   intCounter = intCounter + 1;
   ulngSum = ulngSum + intCounter;
  }
```

To use a `do-while` loop to perform summation, replace LOCs 16 to 18 with the following LOCs:

```
intCounter = 0;
do {
   intCounter = intCounter + 1;
```

```
   ulngSum = ulngSum + intCounter;
  } while (intCounter < 100);
```

Be careful while coding the terminating condition of a loop. An imprudently coded termination condition of a loop is the birthplace of bugs.

Bugs loiter around boundary values.

For example, look at the `for` loop given here:

```
for (intCounter = 1; intCounter < 100; intCounter++) {
       /*  some code here */
  }
```

At first glance, you may think this `for` loop performs 100 iterations, but in reality, it performs only 99 iterations. Therefore, be cautious when dealing with boundary values.

An error in a source code is a *bug*. The process of spotting and correcting the error in source code is called *debugging*.

An expert programmer creates programs with the least number of possible bugs and also knows how to debug a program. It is possible to write a small program that is absolutely free from bugs, but professional programs that consists of thousands of LOCs are never free from bugs.

2-2. Compute the Factorial of a Number

Problem

You want to develop a program to compute the factorial of a number.

Solution

The factorial of a positive integer `n` is denoted by `n!` and is defined as follows:

```
n! = 1 ×  2 ×  ..... ×  n
```

The factorials of a few numbers are given here:

```
0! = 1                          (by definition)
1! = 1
2! = 1 × 2 = 2
3! = 1 × 2 × 3 = 6
```

Write a C program with the following specifications:

- The program uses a `for` loop to compute the factorial of N.
- The program asks the user to enter the number N (0 < N <= 12). If the user enters the number N outside of this range, then the program asks the user to reenter the number.
- When the computed sum is displayed on the screen, the program asks the user whether he or she wants to compute another factorial or quit.

The Code

The following is the code of the C program written with these specifications. Type the following C program in a text editor and save it in the folder `C:\Code` with the file name `fact.c`:

```
/* This program computes the factorial of number N in an interactive manner. */
                                                                        /* BL */
#include <stdio.h>                                                      /* L1 */
                                                                        /* BL */
main()                                                                  /* L2 */
{                                                                       /* L3 */
 int intN, intCounter, flag;                                            /* L4 */
 unsigned long int ulngFact;                                            /* L5 */
 char ch;                                                               /* L6 */
                                                                        /* BL */
 do {                    /* outer do-while loop begins */               /* L7 */
                                                                        /* BL */
  do {                   /* inner do-while loop begins */               /* L8 */
   flag = 0;                                                            /* L9 */
   printf("Enter a number (0 < N <= 12): ");                            /* L10 */
   scanf("%d", &intN);                                                  /* L11 */
   if ((intN <=0) || (intN > 12))                                       /* L12 */
     flag = 1;                                                          /* L13 */
  } while (flag);        /* inner do-while loop ends */                 /* L14 */
                                                                        /* BL  */
 ulngFact = 1;                                                          /* L15 */
                                                                        /* BL  */
  for (intCounter = 1; intCounter <= intN; intCounter++) {              /* L16 */
   ulngFact = ulngFact * intCounter;                                    /* L17 */
  }                                                                     /* L18 */
                                                                        /* BL  */
  printf("Required factorial is: %lu\n", ulngFact);                     /* L19 */
  printf("Do you want to continue? (Y/N) : ");                          /* L20 */
    scanf(" %c", &ch);                                                  /* L21 */
 } while ((ch == 'y') ||(ch == 'Y')); /* outer do-while loop ends */    /* L22 */
                                                                        /* BL  */
```

```
 printf("Thank you.\n");                                    /* L23 */
 return(0);                                                 /* L24 */
}                                                           /* L25 */
```

Compile and execute this program. A run of this program is given here:

```
Enter a number (0 < N <= 12): 6   ↵
Required factorial is: 720
Do you want to continue? (Y/N) : y   ↵
Enter a number (0 < N <= 12): 20   ↵
Enter a number (0 < N <= 12): 12   ↵
Required factorial is: 479001600
Do you want to continue? (Y/N) : n   ↵
Thank you.
```

How It Works

The `for` loop contained in LOCs 16 to 18 computes the factorial of number N. `do-while` loops with two-level nesting are used in this program. The inner `do-while` loop keeps the user inside the loop as long as the user fails to enter the number N in the specified range. The outer `do-while` loop keeps the user inside the loop as long as the user wants to compute the factorial again. The inner `do-while` loop adds robustness to this program. Besides the `for` loop, you can also use `while` or `do-while` loops to compute the factorial of number N.

2-3. Generate a Fibonacci Sequence

Problem

You want to develop a program to compute the Fibonacci sequence.

Solution

Leonardo Fibonacci (1180 to 1250), also known as Leonardo of Pisa, was an Italian mathematician. He wrote a number of excellent treatises on mathematics, such as *Liber Abaci*, *Practica Geometriae*, *Flos*, and *Liber Quadratorum*. The Fibonacci sequence, named after its inventor and mentioned in *Liber Abaci*, begins with 0 and 1, and every successive term is a sum of the two preceding terms. By definition, the first term is 0, and the second term is 1. The first few terms are listed here:

```
First term              By definition          0
Second term             By definition          1
Third term              0 + 1 =                1
Fourth term             1 + 1 =                2
Fifth term              1 + 2 =                3
Sixth term              2 + 3 =                5
```

The terms in the Fibonacci sequence are also called the *Fibonacci numbers*. A possible routine that can generate Fibonacci numbers is given here in pseudocode:

```
declare four int variables a, b, c, and d
a = 0;                          /* by definition */
b = 1;                          /* by definition */
/* #############  loop begins ################# */
print the values of a and b
c = a + b;                      /* compute the next Fibonacci number */
d = b + c;                      /* compute the next Fibonacci number */
a = c;                          /* reset the value of a */
b = d;                          /* reset the value of b */
/* #############  loop ends  ################## */
```

Write a C program with the following specifications:

- The program uses a `for` loop to compute the Fibonacci numbers.
- The program asks the user to enter the number N (0 < N <= 45). If user enters the number N outside of this range, then the program asks the user to reenter the number. The program then generates N Fibonacci numbers.
- When the computed Fibonacci numbers are displayed on the screen, the program asks the user whether he or she wants to compute another Fibonacci sequence or quit.

The Code

The following is the code of the C program written with these specifications. Type the following text (program) in a C file and save it in the folder `C:\Code` with the file name `fibona.c`:

```
/* This program generates N Fibonacci numbers in interactive manner. */
                                                                    /* BL */
#include <stdio.h>                                                  /* L1 */
                                                                    /* BL */
main()                                                              /* L2 */
{                                                                   /* L3 */
 int intN, intK, flag;                                              /* L4 */
 long int lngA, lngB, lngC, lngD;                                   /* L5 */
 char ch;                                                           /* L6 */
                                                                    /* BL */
 do {                 /* outer do-while loop begins */             /* L7 */
                                                                    /* BL */
  do {                /* inner do-while loop begins */             /* L8 */
   flag = 0;                                                        /* L9 */
   printf("Enter a number (0 < N <= 45): ");                        /* L10 */
   scanf("%d", &intN);                                              /* L11 */
```

```
   if ((intN <=0) || (intN > 45))                                    /* L12 */
     flag = 1;                                                       /* L13 */
  } while (flag);     /* inner do-while loop ends */                 /* L14 */
                                                                      /* BL  */
  lngA = 0;                                                           /* L15 */
  lngB = 1;                                                           /* L16 */
  printf("Fibonacci Sequence:\n");                                    /* L17 */
                                                                      /* BL  */
  for (intK = 1; intK <= intN; intK++) {                              /* L18 */
   printf("%d th term is : %ld\n", ((intK * 2) - 1), lngA);            /* L19 */
   if (((intK *2) - 1) == intN) break;                                /* L20 */
   printf("%d th term is : %ld\n", (intK * 2), lngB);                 /* L21 */
   if ((intK * 2) == intN) break;                                     /* L22 */
   lngC = lngA + lngB;                                                /* L23 */
   lngD = lngB + lngC;                                                /* L24 */
   lngA = lngC;                                                       /* L25 */
   lngB = lngD;                                                       /* L26 */
  }                                                                   /* L27 */
                                                                      /* BL  */
  printf("Do you want to continue? (Y/N) : ");                        /* L28 */
    scanf(" %c", &ch);                                                /* L29 */
 } while ((ch == 'y') || (ch == 'Y')); /* outer do-while loop ends */  /* L30 */
                                                                      /* BL  */
 printf("Thank you.\n");                                              /* L31 */
 return(0);                                                           /* L32 */
}                                                                     /* L33 */
```

Compile and execute this program. A run of this program is given here:

```
Enter a number (0 < N <= 45): 1   ↵
Fibonacci Sequence:
1 th term is : 0
Do you want to continue? (Y/N) : y   ↵
Enter a number (0 < N <= 45): 50   ↵
Enter a number (0 < N <= 45): 6   ↵
Fibonacci Sequence:
1 th term is : 0
2 th term is : 1
3 th term is : 1
4 th term is : 2
5 th term is : 3
6 th term is : 5
Do you want to continue? (Y/N) : n   ↵
Thank you.
```

How It Works

The `for` loop contained in LOCs 18 to 27 does most of the work. The code contained in LOCs 23 to 26 computes the Fibonacci numbers. The code contained in LOC 19 and LOC 21 displays the computed Fibonacci numbers on the screen. `do-while` loops with two-level nesting are used in this program. The inner `do-while` loop keeps the user inside the loop as long as the user fails to enter the number N in the specified range. The outer `do-while` loop keeps the user inside the loop as long as the user wants to compute the Fibonacci numbers again. The inner `do-while` loop adds robustness to this program. Besides the `for` loop, you can also use `while` or `do-while` loops to compute the Fibonacci numbers. The Fibonacci sequence has applications in botany, electrical network theory, searching, and sorting.

2-4. Determine Whether a Given Number Is Prime

Problem

You want to develop a program to determine whether a given number is prime.

Solution

A prime number is a positive whole number that is exactly divisible only by 1 and itself. The first few prime numbers are as follows: 2, 3, 5, 7, 11, 13, 17, 19. All prime numbers are odd numbers except 2. You will develop a program that will determine whether a given number is prime.

When program execution begins, you will be asked to enter a number in the range 2 to 2000000000. Type any integer in this range, and the program will tell you whether that number is prime. Also, enter 0 to terminate the program. Obviously, to find out whether a number N is prime, you must divide it by all numbers from 2 through (N – 1) and check the remainder. Number N is a prime number if the remainder is nonzero in the case of each division; otherwise, it is not a prime number. However, in practice, you will divide the number N by all numbers from 2 through √N (the square root of N) and check the remainder. If N is not exactly divisible by any number from 2 through √N, then certainly it is not divisible by any number from 2 through (N – 1).

A routine that will determine whether a given number `lngN` is prime is given here. Here, `isPrime` is an `int` variable; `lngN`, `lngM`, and `i` are `long int` variables; the value of `lngN` is 3 or more; and `isPrime` is set to 1 (to be interpreted as true).

```
isPrime = 1;                                    /* L1 */
lngM = ceil(sqrt(lngN));                        /* L2 */
for (i = 2; i <= lngM; i++) {                   /* L3 */
  if ((lngN % i) == 0) {                        /* L4 */
    isPrime = 0;                                /* L5 */
    break;                                      /* L6 */
  }                                             /* L7, if statement ends */
}                                               /* L8, for loop ends */
```

In this routine, in LOC 2, by implicit type conversion, the value of `lngN` is converted into the `double` type and then fed to `sqrt()` to compute its square root. The result returned by `sqrt()` is fed to `ceil()` to convert it into a nearest whole number on the higher side. The result returned by `ceil()` is then assigned to `lngM` after implicit type conversion.

Next, `lngN` will be divided by all the numbers from 2 through `intM`. If in all these divisions, the remainder is nonzero, then `lngN` will be a prime number, otherwise not. This is done in the `for` loop that spans LOCs 3 to 8. The actual division is performed in LOC 4, and the remainder is checked for its value (whether zero or not). If the remainder is zero, then LOCs 5 and 6 are executed. In LOC 5, the value of the `int` variable `isPrime` is set to zero. In LOC 6, a `break` statement is executed that terminates the `for` loop. Noting the value of `isPrime`, the result is displayed on the screen. If `isPrime` is 1 (true), then `lngN` is a prime number, and if `isPrime` is 0 (false), then `lngN` is not a prime number.

Write a C program with the following specifications:

- The program uses a `for` loop to check the primeness of a number.
- The program asks the user to enter the number N (2 <= N <= 2000000000) to determine whether that number is prime. If the user enters the number N outside of this range, then the program asks the user to reenter the number. The program then checks the primeness of that number. If the user enters 0, then the program is terminated.

The Code

The following is the code of the C program written with these specifications. Type the following text (program) in a C file and save it in the folder `C:\Code` with the file name `prime.c`:

```
/* This program determines whether a given number is prime or not. */
                                                                    /* BL */
#include <stdio.h>                                                  /* L1 */
#include <math.h>                                                   /* L2 */
                                                                    /* BL */
main()                                                              /* L3 */
{                                                                   /* L4 */
 int flag, isPrime;                                                 /* L5 */
 long int lngN, lngM, i;                                            /* L6 */
                                                                    /* BL */
 do{                                                                /* L7 */
                                                                    /* BL */
  do {                                                              /* L8 */
   flag = 0;                                                        /* L9 */
   printf("Enter 0 to discontinue.\n");                             /* L10 */
   printf("Enter a number N (2 <= N <= 2000000000)\n");             /* L11 */
   printf("to find whether it is prime or not: ");                  /* L12 */
   scanf("%ld", &lngN);                                             /* L13 */
   if (lngN == 0) break;                                            /* L14 */
   if ((lngN < 2) || (lngN > 2000000000))                           /* L15 */
```

```
     flag = 1;                                                    /* L16 */
  } while (flag);                                                 /* L17 */
                                                                  /* BL  */
  if (lngN == 0) break;                                           /* L18 */
                                                                  /* BL  */
  if (lngN == 2) {                                                /* L19 */
    printf("\n2 is a prime number\n\n");                          /* L20 */
    continue;                                                     /* L21 */
  }                                                               /* L22 */
                                                                  /* BL  */
  isPrime = 1;                                                    /* L23 */
  lngM = ceil(sqrt(lngN));                                        /* L24 */
    for (i = 2; i <= lngM; i++) {                                 /* L25 */
     if ((lngN % i) == 0) {                                       /* L26 */
        isPrime = 0;                                              /* L27 */
        break;                                                    /* L28 */
     }                                                            /* L29 */
   }                                                              /* L30 */
                                                                  /* BL  */
   if (isPrime)                                                   /* L31 */
     printf("\n%ld is a prime number\n\n", lngN);                 /* L32 */
   else                                                           /* L33 */
     printf("\n%ld is not a prime number\n\n", lngN);             /* L34 */
                                                                  /* BL  */
 } while (1);                                                     /* L35 */
                                                                  /* BL  */
 printf("\nThank you.\n");                                        /* L36 */
 return(0);                                                       /* L37 */
}                                                                 /* L38 */
```

Compile and execute this program. A run of this program is given here:

```
Enter 0 to discontinue.
Enter a number in the range (2 <= N <= 2000000000)
to find whether it is prime or not: 17   ↵
17 is a prime number
Enter 0 to discontinue.
Enter a number in the range (2 <= N <= 2000000000)
to find whether it is prime or not: 1999999997   ↵
1999999997 is not a prime number
Enter 0 to discontinue.
Enter a number in the range (2 <= N <= 2000000000)
to find whether it is prime or not: 0   ↵
Thank you.
```

How It Works

The `for` loop contained in LOCs 25 to 30 does the most of the work of checking the primeness of the number. The code in LOCs 31 to 34 displays the result. `do-while` loops with two-level nesting are used in this program. The inner `do-while` loop keeps the user inside the loop as long as the user fails to enter a number N in the specified range. The outer `do-while` loop keeps the user inside the loop as long as the user wants to checks the primeness of a new number. The inner `do-while` loop adds robustness to this program. Notice LOC 35, which is reproduced here for your quick reference:

```
} while (1);                                                    /* L35 */
```

It seems that this is an infinite loop because no comparison statement is in the parentheses. However, a provision for termination of the loop is made in LOC 18, which is also reproduced here for your quick reference:

```
If (lngN == 0) break;                                           /* L18 */
```

When the value of `lngN` is zero, the execution of this loop is terminated successfully.

The library functions `ceil()` and `sqrt()` are used in LOC 24, which is also reproduced here for your quick reference:

```
lngM = ceil(sqrt(lngN));                                        /* L24 */
```

The library functions `ceil()` and `sqrt()` are mathematical functions; that's why I have included the header file `math.h` in this program through LOC 2. The term `sqrt` stands for "square root," and the term `ceil` stands for "ceiling," which in turn means upper limit. Here is the generic syntax of a statement that uses the library function `sqrt()`:

```
dblX = sqrt(dblY);
```

Here, `dblY` is an expression that evaluates to a constant of the `double` type, and `dblX` is a variable of the `double` type. The function `sqrt()` computes the square root of `dblY` and returns the result, which is assigned to the variable `dblX`.

The function `ceil()` converts the double value (passed as an argument) into a nearest whole-number value on the higher side and returns the result. Here is the generic syntax of a statement that uses the function `ceil()`:

```
dblX = ceil(dblY);
```

Here, `dblY` is an expression that evaluates to a constant of type `double`, and `dblX` is a `double` variable.

2-5. Compute the Sine Function

Problem

You want to compute the sine of an angle x using the infinite series expansion.

Solution

You want to compute the sine of an angle x using the infinite series expansion. The formula of the infinite series expansion is given here:

```
sin x = x - x3/3! + x5/5! - x7/7! + ...
```

Here, x is in radians, and it takes values in the range -1 <= x <= 1. You can see that the value of successive terms go on, decreasing rapidly. Therefore, it is more than sufficient to include only the first ten terms. If the value of x is 1, then the contribution due to the tenth term is approximately 2E-20, and the contribution due to the 40th term is approximately 1.7E-121.

Write a C program with the following specifications:

- The program uses a `for` loop to compute the sine of an angle x.
- The program asks the user to enter the angle x (-1 <= x <= 1). If user enters the angle x outside of this range, then the program asks the user to reenter the number.
- When the sine of angle x is displayed on the screen, the program asks the user whether he or she wants to compute the sine of another angle or quit.

The Code

The following is the code of the C program written with these specifications. Type the following text (program) in a C file and save it in the folder `C:\Code` with the file name `sine.c`:

```
/* This program computes the sine of an angle where angle X is */
/* in radians and in the range (-1 <= X <= 1). */
                                                                  /* BL */
#include <stdio.h>                                                /* L1 */
                                                                  /* BL */
main()                                                            /* L2 */
{                                                                 /* L3 */
 double dblSine, dblTerm, dblX, dblZ;                             /* L4 */
 int intK, i, flag;                                               /* L5 */
 char ch;                                                         /* L6 */
                                                                  /* BL */
 do {                   /* outer do-while loop begins */          /* L7 */
                                                                  /* BL */
```

```
  do {                    /* inner do-while loop begins */        /* L8 */
   flag = 0;                                                      /* L9 */
   printf("Enter angle in radians (-1 <= X <= 1): ");             /* L10 */
   scanf("%lf", &dblX);                                           /* L11 */
   if ((dblX < -1) || (dblX > 1))                                 /* L12 */
     flag = 1;                                                    /* L13 */
  } while (flag);         /* inner do-while loop ends */          /* L14 */
                                                                  /* BL  */
  dblTerm = dblX;                                                 /* L15 */
  dblSine = dblX;                                                 /* L16 */
  intK = 1;                                                       /* L17 */
  dblZ = dblX * dblX;                                             /* L18 */
                                                                  /* BL  */
  for (i = 1; i <= 10; i++) {                                     /* L19 */
   intK = intK + 2;                                               /* L20 */
   dblTerm = -dblTerm * dblZ /(intK * (intK - 1));                /* L21 */
   dblSine = dblSine + dblTerm;                                   /* L22 */
  }                                                               /* L23 */
                                                                  /* BL  */
  printf("Sine of %lf is %lf\n", dblX, dblSine);                  /* L24 */
  printf("Do you want to continue? (Y/N) : ");                    /* L25 */
   scanf(" %c", &ch);                                             /* L26 */
 } while ((ch == 'y') || (ch == 'Y')); /* outer do-while ends */   /* L27 */
                                                                  /* BL */
 printf("Thank you.\n");                                          /* L28 */
 return(0);                                                       /* L29 */
}                                                                 /* L30 */
```

Compile and execute this program. A run of this program is given here:

```
Enter angle in radians (-1 <= X <= 1): 0.5   ↵
Sine of 0.500000 is 0.479426
Do you want to continue? (Y/N) : y   ↵
Enter angle in radians (-1 <= X <= 1): 0   ↵
Sine of 0.000000 is 0.000000
Do you want to continue? (Y/N) : y   ↵
Enter angle in radians (-1 <= X <= 1): 0.707   ↵
Sine of 0.707000 is 0.649556
Do you want to continue? (Y/N) : n   ↵
Thank you.
```

How It Works

The `for` loop contained in LOCs 19 to 23 computes the sine of an angle `x`. The code in LOC 24 displays the result. `do-while` loops with two-level nesting are used in this program. The inner `do-while` loop keeps the user inside the loop as long as the user fails to enter the angle `x` in the specified range. The outer `do-while` loop keeps the user inside the loop as long as the user wants to compute the sine of another angle. The inner `do-while` loop adds robustness to this program.

2-6. Compute the Cosine Function

Problem

You want to compute the cosine of an angle x using the infinite series expansion.

Solution

You want to compute the cosine of an angle x using the infinite series expansion. The formula of the infinite series expansion is given here:

```
cos x = 1 - x2/2! + x4/4! - x6/6! +...
```

Here, x is in radians, and it takes values in the range -1 <= x <= 1. You can see that the value of successive terms go on, decreasing rapidly. Therefore, it is more than sufficient to include only the first ten terms, as discussed in the preceding recipe.

Write a C program with the following specifications:

- The program uses a `for` loop to compute the cosine of an angle x.
- The program asks the user to enter the angle x (-1 <= x <= 1). If user enters the angle x outside of this range, then the program asks the user to reenter the number.
- When the cosine of angle x is displayed on the screen, the program asks the user whether he or she wants to compute the cosine of another angle or quit.

The Code

The following is the code of the C program written with these specifications. This time, however, you use a slightly different algorithm for coding compared to the preceding recipe. Type the following text (program) in a C file and save it in the folder `C:\Code` with the file name `cosine.c`:

```
/* This program computes the cosine of an angle where angle X is */
/* in radians and in the range (-1 <= X <= 1). */
                                                                  /* BL */
#include <stdio.h>                                                /* L1 */
                                                                  /* BL */
main()                                                            /* L2 */
{                                                                 /* L3 */
 double dblCosine, dblX, dblZ;                                    /* L4 */
 int i, j, q, flag, factorial, sign;                              /* L5 */
 char ch;                                                         /* L6 */
                                                                  /* BL */
 do {                    /* outer do-while loop begins */         /* L7 */
                                                                  /* BL */
  do {                   /* inner do-while loop begins */         /* L8 */
```

```
   flag = 0;                                                    /* L9 */
   printf("Enter angle in radians (-1 <= X <= 1): ");           /* L10 */
   scanf("%lf", &dblX);                                         /* L11 */
   if ((dblX < -1) || (dblX > 1))                               /* L12 */
     flag = 1;                                                  /* L13 */
  } while (flag);        /* inner do-while loop ends */        /* L14 */
                                                                /* BL  */
dblCosine = 0;                                                  /* L15 */
sign = -1;                                                      /* L16 */
for (i = 2; i <= 10; i += 2)                                    /* L17 */
    {                                                           /* L18 */
        dblZ = 1;                                               /* L19 */
        factorial = 1;                                          /* L20 */
                                                                /* BL  */
        for (j = 1; j <= i; j++)                                /* L21 */
                                                                /* L22 */
            dblZ = dblZ * dblX;                                 /* L23 */
            factorial = factorial * j;                          /* L24 */
                                                                /* L25 */
                                                                /* BL  */
        dblCosine += sign * dblZ / factorial;                   /* L26 */
        sign =  - 1 * sign;                                     /* L27 */
    }                                                           /* L28 */
  dblCosine = 1 + dblCosine;                                    /* L29 */
                                                                /* BL  */
  printf("Cosine of %lf is %lf\n", dblX, dblCosine);            /* L30 */
  printf("Do you want to continue? (Y/N) : ");                  /* L31 */
   scanf(" %c", &ch);                                           /* L32 */
 } while ((ch == 'y') || (ch == 'Y')); /* outer do-while ends */ /* L33 */
                                                                /* BL */
 printf("Thank you.\n");                                        /* L34 */
 return(0);                                                     /* L35 */
}                                                               /* L36 */
```

Compile and execute this program. A run of this program is given here:

```
Enter angle in radians (-1 <= X <= 1): 0.5   ↵
Cosine of 0.500000 is 0.8775826
Do you want to continue? (Y/N) : y   ↵
Enter angle in radians (-1 <= X <= 1): 0   ↵
Cosine of 0.000000 is 1.000000
Do you want to continue? (Y/N) : y   ↵
Enter angle in radians (-1 <= X <= 1): 0.707   ↵
Cosine of 0.707000 is 0.760309
Do you want to continue? (Y/N) : n   ↵
Thank you.
```

How It Works

Two-level nesting of `for` loops is used to compute the cosine of an angle `x`. LOC 30 displays the result. `do-while` loops with two-level nesting are used in this program. The inner `do-while` loop keeps the user inside the loop as long as the user fails to enter the angle `x` in the specified range. The outer `do-while` loop keeps the user inside the loop as long as the user wants to compute the cosine of another angle. The inner `do-while` loop adds robustness to this program.

2-7. Compute the Roots of Quadratic Equation

Problem

You want to compute the roots of the quadratic equation.

Solution

You want to compute the roots of the quadratic equation ax2 + bx + c = 0. These roots are given by the following formulae:

```
(-b + √(b2 - 4ac))/2a        and      (-b - √(b2 - 4ac))/2a
```

Roots can be real or imaginary depending upon the values of `a`, `b`, and `c`. Write a C program with the following specifications:

- The program asks the user to enter the values of `a`, `b`, and `c`, which can be integers or floating-point numbers.
- The program computes the roots and displays the results on the screen using the formulae given earlier.
- When roots of a quadratic equation are displayed on the screen, the program asks the user whether he or she wants to compute the roots of another quadratic equation or quit.

The Code

The following is the code of the C program written with these specifications. Type the following text (program) in a C file and save it in the folder `C:\Code` with the file name `roots.c`:

```
/* This program computes the roots of quadratic equation. */
                                                                    /* BL */
#include <stdio.h>                                                  /* L1 */
#include <math.h>                                                   /* L2 */
                                                                    /* BL */
main()                                                              /* L3 */
{                                                                   /* L4 */
```

```
    double dblA, dblB, dblC, dblD, dblRt1, dblRt2;              /* L5 */
    char ch;                                                    /* L6 */
    do {                /* do-while loop begins */              /* L7 */
                                                                /* BL */
    printf("Enter the values of a, b and c : ");                /* L8 */
    scanf("%lf %lf %lf", &dblA, &dblB, &dblC);                  /* L9 */
                                                                /* BL */
    dblD = dblB * dblB - 4 * dblA * dblC;                       /* L10 */
    if (dblD == 0)                                              /* L11 */
    {                                                           /* L12 */
        dblRt1 = ( - dblB) / (2 * dblA);                        /* L13 */
        dblRt2 = dblRt1;                                        /* L14 */
        printf("Roots are real & equal\n");                     /* L15 */
        printf("Root1 = %f, Root2 = %f\n", dblRt1, dblRt2);     /* L16 */
    }                                                           /* L17 */
    else if (dblD > 0)                                          /* L18 */
    {                                                           /* L19 */
        dblRt1 =  - (dblB + sqrt(dblD)) / (2 * dblA);           /* L20 */
        dblRt2 =  - (dblB - sqrt(dblD)) / (2 * dblA);           /* L21 */
        printf("Roots are real & distinct\n");                  /* L22 */
        printf("Root1 = %f, Root2 = %f\n", dblRt1, dblRt2);     /* L23 */
    }                                                           /* L24 */
    else                                                        /* L25 */
    {                                                           /* L26 */
        printf("Roots are imaginary\n");                        /* L27 */
    }                                                           /* L28 */
   printf("Do you want to continue? (Y/N) : ");                 /* L29 */
   scanf(" %c", &ch);                                           /* L30 */
 } while ((ch == 'y') || (ch == 'Y')); /* do-while loop ends */ /* L31 */
   printf("Thank you.\n");                                      /* L32 */
                                                                /* BL  */
   return 0;                                                    /* L33 */
}                                                               /* L34 */
```

Compile and execute this program. A run of this program is given here:

```
Enter the values of a, b and c : 10  200  -30   ↵
Roots are real and distinct
Root1 = -20.148892,  Root2 = 0.148892
Do you want to continue? (Y/N) : y   ↵
Enter the values of a, b and c : 40  20  15   ↵
Roots are imaginary
Do you want to continue? (Y/N) : n   ↵
Thank you.
```

How It Works

Simple mathematical operations are performed to compute the roots of the quadratic equation. Roots can be real or imaginary depending upon the values of the coefficients `a`, `b`, and `c`. Therefore, provision is made to test whether the roots are real or imaginary. The `do-while` loop keeps the user inside the loop as long as the user wants to compute the roots of another quadratic equation.

2-8. Compute the Reverse of an Integer

Problem

You want to compute the reverse of an integer.

Solution

You want to compute the reverse of an integer. For example, if a given integer is 12345, then its reverse is 54321, and you want to compute it programatically.

Write a C program with the following specifications:

- The program asks the user to enter an integer N (0 < N <= 30000). If the user enters the integer N outside of this range, then the program asks the user to reenter the integer.
- The program computes the reverse of an integer and displays the results on the screen.
- Then the program asks the user whether he or she wants to compute the reverse of another integer or quit.

The Code

The following is the code of the C program written with these specifications. Type the following text (program) in a C file and save it in the folder `C:\Code` with the file name `reverse.c`:

```
/* This program computes the reverse of an integer number. */
                                                                    /* BL */
#include <stdio.h>                                                  /* L1 */
                                                                    /* BL */
main()                                                              /* L2 */
{                                                                   /* L3 */
    long int intN, intTemp, intRemainder, intReverse;               /* L4 */
    char ch;                                                        /* L5 */
    do {                    /* outer do-while loop begins */        /* L6 */
      do {                  /* inner do-while loop begins */        /* L7 */
        printf("Enter a number (0 < N <= 30000): ");                /* L8 */
        scanf("%ld", &intN);                                        /* L9 */
```

```
      } while ((intN <= 0) || (intN > 30000));
        /* inner do-while loop ends */                          /* L10 */
                                                                /* BL  */
      intTemp = intN;                                           /* L11 */
      intReverse = 0;                                           /* L12 */
                                                                /* BL3 */
      while (intTemp > 0)                                       /* L14 */
       {                                                        /* L15 */
         intRemainder = intTemp % 10;                           /* L16 */
         intReverse = intReverse * 10 + intRemainder;           /* L17 */
         intTemp /= 10;                                         /* L18 */
       }                                                        /* L19 */
                                                                /* BL  */
      printf("The reverse of %ld is %ld.\n", intN, intReverse); /* L20 */
      printf("Do you want to continue? (Y/N) : ");              /* L21 */
      scanf(" %c", &ch);                                        /* L22 */
    } while ((ch == 'y') || (ch == 'Y'));
      /* outer do-while loop ends */                            /* L23 */
    printf("Thank you.\n");                                     /* L24 */
                                                                /* BL  */
    return 0;                                                   /* L25 */
}                                                               /* L26 */
```

Compile and execute this program. A run of this program is given here:

```
Enter a number (0 < N <= 30000): 12345   ↵
The reverse of 12345 is 54321.
Do you want to continue? (Y/N): y   ↵
Enter a number (0 < N <= 30000): 45678   ↵
Enter a number (0 < N <= 30000): 2593   ↵
The reverse of 2593 is 3952.
Do you want to continue? (Y/N): n   ↵
Thank you.
```

How It Works

Simple mathematical operations are performed to compute the reverse of an integer. `do-while` loops with two-level nesting are used in this program. The inner `do-while` loop keeps the user inside the loop as long as the user fails to enter the integer N in the specified range. The outer `do-while` loop keeps the user inside the loop as long as the user wants to compute the reverse of another integer. The inner `do-while` loop adds robustness to this program.

2-9. Print a Geometrical Pattern Using Nested Loops

Problem

You want to generate and print on-screen the following geometrical pattern using the nested loops (and not using the naïve five `printf()` statements):

```
    1
   212
  32123
 4321234
543212345
```

The order of this pattern is five; i.e., it consists of five lines. You want to generate the pattern of any order from one to nine.

Solution

You can print this pattern programatically using two-level nesting of `for` loops. Write a C program with the following specifications:

- The program asks the user to enter the order of the pattern (1 <= N <= 9). If the user enters an N outside of this range, then the program asks the user to reenter the N.
- The program prints the desired pattern using two-level nesting of `for` loops. However, there will be four `for` loops in this program.
- Then the program asks the user whether he or she wants to print another pattern or quit.

The Code

The following is the code of the C program written with these specifications. Type the following text (program) in a C file and save it in the folder `C:\Code` with the file name `pattern.c`:

```
/* This program prints the geometrical pattern on the screen. */
                                                                    /* BL */
#include <stdio.h>                                                  /* L1 */
                                                                    /* BL */
main()                                                              /* L2 */
{                                                                   /* L3 */
    int intI, intJ, intK, intL, intOrd;                             /* L4 */
    char ch;                                                        /* L5 */
    do {                     /* do-while loop begins */             /* L6 */
      do {                  /* do-while loop begins */             /* L7 */
```

```
        printf("Enter the order of pattern (0 < N < 10): ");       /* L8 */
        scanf("%d", &intOrd);                                      /* L9 */
      } while ((intOrd <= 0) || (intOrd >= 10));
        /* do-while loop ends */                                   /* L10 */
                                                                   /* BL  */
    for (intI = 1; intI <= intOrd; intI++)                         /* L11 */
      {                                                            /* L12 */
        for (intJ = intOrd; intJ > intI; intJ--)                   /* L13 */
        {                                                          /* L14 */
            printf(" ");                                           /* L15 */
        }                                                          /* L16 */
        for (intK = intI; intK >= 1; intK--)                       /* L17 */
        {                                                          /* L18 */
            printf("%d", intK);                                    /* L19 */
        }                                                          /* L20 */
        for (intL = 2; intL <= intI; intL++)                       /* L21 */
        {                                                          /* L22 */
            printf("%d", intL);                                    /* L23 */
        }                                                          /* L24 */
        printf("\n");                                              /* L25 */
      }                                                            /* L26 */
      printf("Do you want to continue? (Y/N) : ");                 /* L27 */
      scanf(" %c", &ch);                                           /* L28 */
    } while ((ch == 'y') || (ch == 'Y')); /* do-while loop ends */ /* L29 */
    printf("Thank you\n");                                         /* L30 */
                                                                   /* BL  */
    return 0;                                                      /* L31 */
}                                                                  /* L32 */
```

Compile and execute this program. A run of this program is given here:

```
Enter the order of pattern (0 < N < 10): 5   ↵
    1
   212
  32123
 4321234
543212345
Do you want to continue? (Y/N) : n   ↵
Thank you.
```

How It Works

A correct combination of `for` loops generates the desired pattern. `do-while` loops with two-level nesting are used in this program. The inner `do-while` loop keeps the user inside the loop as long as the user fails to enter the integer N in the specified range. The outer `do-while` loop keeps the user inside the loop as long as the user wants to generate another pattern of a different order. The inner `do-while` loop adds robustness to this program.

2-10. Generate a Table of Future Value Interest Factors

Problem

You want to generate a table of future value interest factors (FVIFs) and print it on the screen.

Solution

You can generate and print this table programmatically using two-level nesting of `for` loops. Write a C program with the following specifications:

- The program generates the FVIF table for interest rates varying from 1 percent to 6 percent and for the periods varying from 1 year to 10 years.
- The FVIF values should be accurate up to three decimal points.

The Code

The following is the code of the C program written with these specifications. Type the following text (program) in a C file and save it in the folder `C:\Code` with the file name `interest.c`:

```
/* This program computes table of FVIF, Future Value Interest Factors. */
                                                                        /* BL */
#include <stdio.h>                                                      /* L1 */
#include <math.h>                                                       /* L2 */
                                                                        /* BL */
#define  MAX_INTEREST  6                                                /* L3 */
#define  MAX_PERIOD  10                                                 /* L4 */
                                                                        /* BL */
main()                                                                  /* L5 */
{                                                                       /* L6 */
  int i, interest, years;                                               /* L7 */
  float fvif;                                                           /* L8 */
  printf("\nTable of FVIF (Future Value Interest Factors).");           /* L9 */
  printf("\nRate of interest varies from 1% to 6%.");                   /* L10 */
  printf("\nPeriod varies from 1 year to 10 years.");                   /* L11 */
  printf("\n\n                    Interest Rate  ");                   /* L12 */
  printf("\n\t ------------------------------------------");           /* L13 */
  printf("\nPeriod");                                                   /* L14 */
  for (i=1; i<= MAX_INTEREST; i++)                                      /* L15 */
    printf("\t   %d%", i);                                             /* L16 */
  printf("\n-------------------------------------------------\n");     /* L17 */
  for(years=1; years <= MAX_PERIOD; years++) {                          /* L18 */
```

```
    printf("%d\t", years);                                          /* L19 */
    for(interest=1; interest <= MAX_INTEREST; interest++) {         /* L20 */
      fvif = pow((1+interest*0.01), years);                         /* L21 */
      printf("%6.3f\t", fvif);                                      /* L22 */
    }                                                               /* L23 */
    printf("\n");                                                   /* L24 */
  }                                                                 /* L25 */
  printf("-------------------------------------------------\n");    /* L26 */
  printf("Thank you.\n");                                           /* L27 */
  return 0;                                                         /* L28 */
}                                                                   /* L29 */
```

Compile and execute this program. A run of this program is given here:

```
Table of FVIF (Future Value Interest Factors).
Rate of Interest varies from 1% to 6%.
Period varies from 1 year to 10 years.

                        Interest Rate
          ---------------------------------------------
Period      1%      2%      3%      4%      5%      6%
-----------------------------------------------------------
1         1.010   1.020   1.030   1.040   1.050   1.060
2         1.020   1.040   1.061   1.082   1.102   1.124
3         1.030   1.061   1.093   1.125   1.158   1.191
4         1.041   1.082   1.126   1.170   1.216   1.262
5         1.051   1.104   1.159   1.217   1.276   1.338
6         1.062   1.126   1.194   1.265   1.340   1.419
7         1.072   1.149   1.230   1.316   1.407   1.504
8         1.083   1.172   1.267   1.369   1.477   1.594
9         1.094   1.195   1.305   1.423   1.551   1.689
10        1.105   1.219   1.344   1.480   1.629   1.791
-----------------------------------------------------------
Thank you.
```

How It Works

FVIF tables are useful in calculating the future value of money. The future value (FVn) of the principal amount (P_0) after n years, with a rate of interest (i%) per annum, is given by the following formulae:

$$FV_n = P_0 * (1 + i)n = P_0 * FVIF$$

Here, FVIF = $(1 + i)_n$.

Suppose the principal amount is US$200, the rate of interest is 6 percent per annum, and the period is 8 years; from the previous table you can find that the corresponding FVIF is 1.594 (last column, eighth row). The future value for this amount is given by the following:

$$FV_n = US\$200 * 1.594 = US\$318.80$$

In this program, LOCs 1 to 2 consist of `include` statements. LOCs 3 to 4 consist of `define` statements. LOCs 5 to 29 consist of the definition of the `main()` function. In LOCs 7 to 8, a few variables are declared. LOCs 9 to 11 consist of three `printf()` statements that print the information about the FVIF table. LOCs 12 to 17 print the heading of the FVIF table. LOCs 18 to 25 consist of nested `for` loops. The FVIF table is calculated and printed in these nested loops. LOC 26 prints the bottom line of the FVIF table.