3

Structured Program Development in C

4.2 Repetition Essentials

- **Loop**
  - **Group of instructions computer executes repeatedly while some condition remains `true`**
- **Sentinel-controlled repetition**
  - **Indefinite repetition**
  - **Used when number of repetitions not known**
  - **Sentinel value indicates "end of data"**
- **Counter-controlled repetition**
  - **Definite repetition: know how many times loop will execute**
  - **Control variable used to count repetitions**

3.7 The `while` repetition statement

- **Repetition structure**
  - **Programmer specifies an action to be repeated while some condition remains `true`**
  - **Pseudocode:**

    While there are more items on my shopping list
    Purchase next item and cross it off my list
  - **`while` loop repeated until condition becomes `false`**
  - **Example:**

```
int product = 2;
while ( product <= 1000 )
      product = 2 * product;
```

Common Programming Error 3.3

Not providing the body of a `while` statement with an action that eventually causes the condition in the `while` to become false. Normally, such a repetition structure will never terminate—an error called an "infinite loop."

Program Example: fig03_08.c

Use a special sentinel value to indicate "end of data entry"

→ loop ends when user inputs the sentinel value

In a sentinel-controlled loop, the prompts requesting data entry should explicitly remind the user what the sentinel value is.

```
23    /* loop while sentinel value not yet read from user */
24    while ( grade != -1 ) {
25       total = total + grade; /* add grade to total */
26       counter = counter + 1; /* increment counter */
27
28       /* get next grade from user */
29       printf( "Enter grade, -1 to end: " ); /* prompt for input */
30       scanf("%d", &grade);                  /* read next grade */
31    } /* end while */
32
33    /* termination phase */
34    /* if user entered at least one grade */
35    if ( counter != 0 ) {
36
37       /* calculate average of all grades entered */
38       average = ( float ) total / counter; /* avoid truncation */
39
40       /* display average with two digits of precision */
41       printf( "Class average is %.2f\n", average );
42    } /* end if */
43    else { /* if no grades were entered, output message */
44       printf( "No grades were entered\n" );
45    } /* end else */
```

4

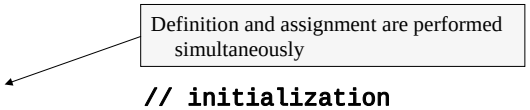
C Program Control

4.3 Counter-Controlled Repetition

- **Counter-controlled repetition**
  - **Loop repeated until counter reaches a certain value**
  - **Definite repetition: number of repetitions is known**
- **Required items:**
  - **The name of a control variable (or loop counter)**
  - **The initial value of the control variable**
  - **An increment (or decrement) by which the control variable is modified each time through the loop**
  - **A condition that tests for the final value of the control variable (i.e., whether looping should continue)**

4.3 Counter-Controlled Repetition

- **Example:**




```
int counter = 1;           // initialization
while ( counter <= 10 ) { // repetition condition
   printf( "%d\n", counter );
   ++counter;              // increment
}
```

  - **The statement**

```
int counter = 1;
```

    - **Names `counter`**
    - **Defines it to be an integer**
    - **Reserves space for it in memory**
    - **Sets it to an initial value of `1`**

4.3 Counter-Controlled Repetition

- **Condensed code**
  - **C Programmers would make the program more concise**
  - **Initialize `counter` to `0`**
    - `while ( ++counter <= 10 )`
      `printf( "%d\n, counter );`

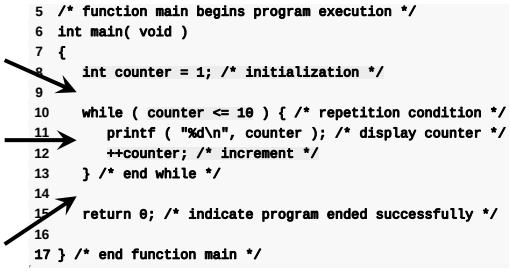
Common Programming Error 4.1

Because floating-point values may be approximate, controlling counting loops with floating-point variables may result in imprecise counter values and inaccurate tests for termination.

→ Control counting loops with integer values.

Good Programming Practice

4.1: Indent the statements in the body of each control statement.



```
5  /* function main begins program execution */
6  int main( void )
7  {
8     int counter = 1; /* initialization */
9
10    while ( counter <= 10 ) { /* repetition condition */
11       printf ( "%d\n", counter ); /* display counter */
12       ++counter; /* increment */
13    } /* end while */
14
15    return 0; /* indicate program ended successfully */
16
17 } /* end function main */
```

4.2: Put a blank line before and after each control statement to make it stand out in a program.

Good Programming Practice 4.3

Too many levels of nesting can make a program difficult to understand. As a general rule, try to avoid using more than three levels of nesting.

4.4 `for` Repetition Statement

- **Format when using `for` loops**

  `for` (*initialization*; *loopContinuationTest*; *increment*)
  statement

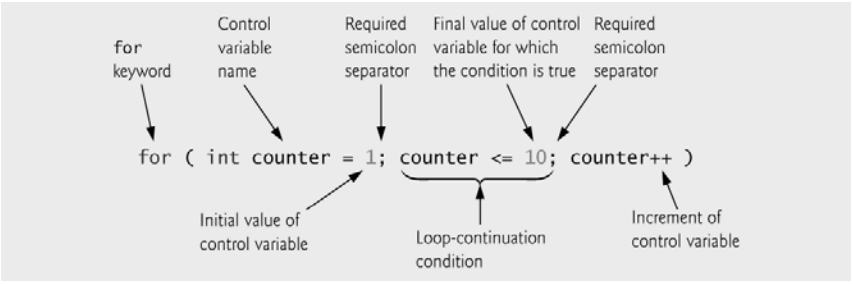


Fig. 4.3 | **`for`** statement header components.

Outline

fig04_02.c

```
1  /* Fig. 4.2: fig04_02.c
2     Counter-controlled repetition with the for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     int counter; /* define counter */
9
10    /* initialization, repetition condition, and increment
11       are all included in the for statement header. */
12    for ( counter = 1; counter <= 10; counter++ ) {
13       printf( "%d\n", counter );
14    } /* end for */
15
16    return 0; /* indicate program ended successfully */
17
18 } /* end function main */
```

`for` loop begins by setting **`counter`** to 1 and repeats while **`counter <= 10`**. Each time the end of the loop is reached, **`counter`** is incremented by 1.

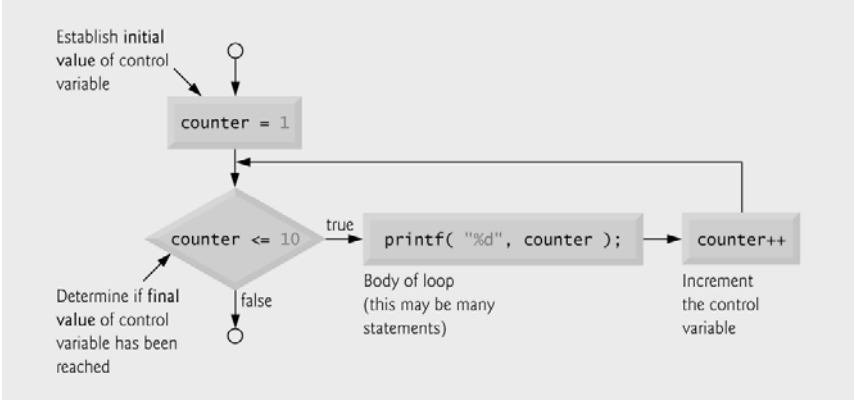


Fig. 4.4 | Flowcharting a typical `for` repetition statement.

Error-Prevention Tip 4.2

Using the final value in the condition of a `while` or `for` statement and using the `<=` relational operator will help avoid off-by-one errors. For a loop used to print the values 1 to 10, for example, the loop-continuation condition should be `counter <= 10` rather than `counter < 11` or `counter < 10`.

4.4 `for` Repetition Statement

- For loops can usually be rewritten as while loops:

```
initialization;
while ( loopContinuationTest ) {
  statement;
  increment;
}
```

- Initialization and increment
  - Can be comma-separated lists
  - Example:

```
for (int i = 0, j = 0;  j + i <= 10; j++, i++)
   printf( "%d\n", j + i );
```

Common Programming Error 4.4

Placing a semicolon immediately to the right of a `for` header makes the body of that `for` statement an empty statement. This is normally a logic error.

```
for ( counter=1; counter<=10; counter++ );
```

→ Do nothing for 10 times

4.5 `for` Statement : Notes and Observations

- **Arithmetic expressions**
  - **Initialization, loop-continuation, and increment can contain arithmetic expressions. If `x` equals `2` and `y` equals `10`**

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

  is equivalent to

```
for ( j = 2; j <= 80; j += 5 )
```

- **Notes about the `for` statement:**
  - **"Increment" may be negative (decrement)**
  - **If the loop continuation condition is initially `false`**
    - **The body of the `for` statement is not performed**
    - **Control proceeds with the next statement after the `for` statement**
  - **Control variable**
    - **Often printed or used inside for body, but not necessary**

Error-Prevention Tip 4.3

Although the value of the control variable can be changed in the body of a `for` loop, this can lead to subtle errors. It is best not to change it.

Outline

fig04_06.c

(1 of 2)

```
1  /* Fig. 4.6: fig04_06.c
2     Calculating compound interest */
3  #include <stdio.h>
4  #include <math.h>
5
6  /* function main begins program execution */
7  int main( void )
8  {
9     double amount;             /* amount on deposit */
10    double principal = 1000.0; /* starting principal */
11    double rate = .05;         /* annual interest rate */
12    int year;                  /* year counter */
13
14    /* output table column head */
15    printf( "%4s%21s\n", "Year", "Amount on deposit" );
16
17    /* calculate amount on deposit for each of ten years */
18    for ( year = 1; year <= 10; year++ ) {
19
20       /* calculate new amount for specified year */
21       amount = principal * pow( 1.0 + rate, year );
22
23       /* output one table row */
24       printf( "%4d%21.2f\n", year, amount );
25    } /* end for */
26
27    return 0; /* indicate program ended successfully */
28
29 } /* end function main */
```

additional header

pow function calculates the value of the first argument raised to the power of the second argument

Outline

fig04_06.c

(2 of 2)

```
Year    Amount on deposit
   1             1050.00
   2             1102.50
   3             1157.63
   4             1215.51
   5             1276.28
   6             1340.10
   7             1407.10
   8             1477.46
   9             1551.33
  10             1628.89
```

4.8 do…while Repetition Statement

- The do…while repetition statement
  - Similar to the while structure
  - Condition for repetition only tested after the body of the loop is performed
    - All actions are performed at least once
  - Format:

```
do {
   statement;
} while ( condition );
```

4.8 do…while Repetition Statement

- Example (letting counter = 1):

```
do {
   printf( "%d  ", counter );
} while (++counter <= 10);
```

  - Prints the integers from 1 to 10

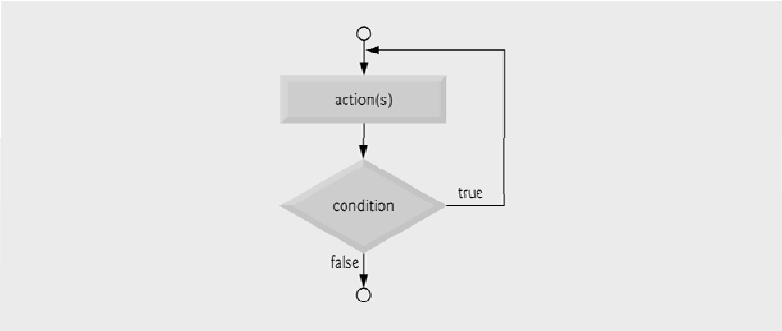


Fig. 4.10 | Flowcharting the **`do...while`** repetition statement.

Good Programming Practice 4.10

Some programmers always include braces in a `do...while` statement even if the braces are not necessary. This helps eliminate ambiguity between the `do...while` statement containing one statement and the `while` statement.

while v.s. do...while

```
total = 0; grade = 0;
counter = 0;
printf("Enter grade, ");
printf("-1 to end: ");
scanf("%d", &grade);
while (grade != -1) {
  total = total + grade;
  counter = counter + 1;
  printf("Enter grade, ");
  printf("-1 to end: ");
  scanf("%d", &grade);
}
```

- Duplicate input statements !!
- Initial values are all zero.

```
total = 0; grade = 0;
counter = -1;
do {
  total = total + grade;
  counter = counter + 1;
  printf("Enter grade, ");
  printf("-1 to end: ");
  scanf("%d", &grade);
} While (grade != -1);
```

- No duplicate input statements !!
- Initial counter starts from -1.

4.9 break and continue Statements

- break
  - Causes immediate exit from a while, for, do...while or switch statement
  - Program execution continues with the first statement after the structure
  - Common uses of the break statement
    - Escape early from a loop
    - Skip the remainder of a switch statement

Outline

fig04_11.c

```
1  /* Fig. 4.11: fig04_11.c
2     Using the break statement in a for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     int x; /* counter */
9
10    /* loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13       /* if x is 5, terminate loop */
14       if ( x == 5 ) {
15          break; /* break loop only if x is 5 */
16       } /* end if */
17
18       printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nBroke out of loop at x == %d\n", x );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */
```

break immediately ends **for** loop

```
1 2 3 4
Broke out of loop at x == 5
```

4.9 `break` and `continue` Statements

- `continue`
  - Skips the remaining statements in the body of a `while`, `for` or `do…while` statement
    - Proceeds with the next iteration of the loop
  - `while` and `do…while`
    - Loop-continuation test is evaluated immediately after the `continue` statement is executed
  - `for`
    - Increment expression is executed, then the loop-continuation test is evaluated

Outline

fig04_12.c

```
1  /* Fig. 4.12: fig04_12.c
2     Using the continue statement in a for statement */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     int x; /* counter */
9
10    /* loop 10 times */
11    for ( x = 1; x <= 10; x++ ) {
12
13       /* if x is 5, continue with next iteration of loop */
14       if ( x == 5 ) {
15          continue; /* skip remaining code in loop body */
16       } /* end if */
17
18       printf( "%d ", x ); /* display value of x */
19    } /* end for */
20
21    printf( "\nUsed continue to skip printing the value 5\n" );
22
23    return 0; /* indicate program ended successfully */
24
25 } /* end function main */
```

```
 1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

`continue` skips to end of **`for`** loop and performs next iteration

Common Programming Error 4.7

Infinite loops are caused when the loop-continuation condition in a `while`, `for` or `do...while` statement never becomes false. To prevent this, make sure there is not a semicolon immediately after the header of a `while` or `for` statement. In a counter-controlled loop, make sure the control variable is incremented (or decremented) in the loop. In a sentinel-controlled loop, make sure the sentinel value is eventually input.

Usage of Infinite Loops

- **Infinite loops are helpful when the termination condition is generated inside the loop**

```
while (1) {
   ......
   ans = a * b;
   if (ans == 0) break;
   ......
}
```

1 (non-zero value) means always TRUE

- **Should be used with *break* to terminate the loop**
  - `Make sure the condition will eventually become TRUE`
- **If sentinel-controlled loop can be used instead, use it !!**
  - `Infinite loops are not easy to debug`

Appendix: Nested Loops

- **Nested loops (loop inside a loop) are allowed in C/C++**

```
for (i=0; i<n; i++) {
    ......
    for (j=0; j<m; j++)
    {  ......  }
}
```

outer loop: run inner loop for n times

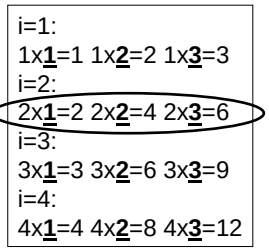
inner loop: repeated actions

- **Similar to migrating 1-dimensional problems into multi-dimensional problems**
  - **One loop: f(0), f(1), f(2), …**
  - **Two loops: f(0,0), f(0,1), …, f(0,n), f(1,0), f(1,1), …**
- **The most important thing:**
  - **Obtain the changing rules of the running index**

Nested Loops: Examples (1/3)

- **Execute multi-dimensional operations**

```
for (i=1; i<=4; i++) {
   printf("i=%d:\n",i);
   for (j=1; j<=3; j++) {
      printf("%dx%d=%d ",i,j,i*j);
   }
   printf("\n");
}
```



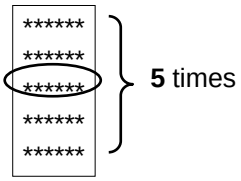
- **Please pay special attention to the index changing sequence**
  - Column first in this case
    (1,1) -> (1,2) -> (1,3) -> (2,1) -> ...
  - So, column is changed in the inner loop

Nested Loops: Examples (2/3)

- **Repeat a loop for n times**

```
for (i=1; i<=5; i++) {
   for (j=1; j<=6; j++)
   {  printf("*");  }
   printf("\n");
}
```

6 stars



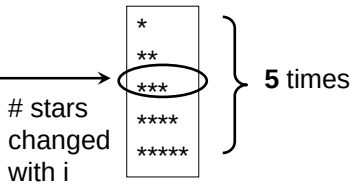
5 times

- **Inner loop control the repeated actions**
  - Print 6 stars in this case
- **Outer loop control the number of times**
  - Print 5 rows of stars in this case

Nested Loops: Examples (3/3)

- **Inner loop is controlled by outer loop**

```
for (i=1; i<=5; i++)
{  for (j=1; j<=i; j++)
   {  printf("*");  }
   printf("\n");
}
```

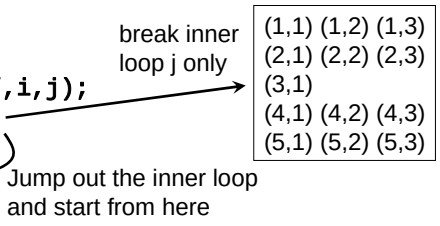


- **Outer loop control the number of rows**
  - `Print 5 rows of stars in this case`
- **Inner loop control the number of stars**
  - `Change its termination condition`
  - `i=1 --> for (j=1;j<=1;j++) --> 1 star`
  - `i=2 --> for (j=1;j<=2;j++) --> 2 stars`
  - `......`

`break` in Nested Loops

- **In nested loops, *`break/continue`* can only affect the most inner loop where the *`break/continue`* stands**

```
for (i=1; i<=5; i++)
{  for (j=1; j<=3; j++)
   {  printf("(%d,%d) ",i,j);
      if (i==3) break;
   }
   printf("\n");
}
```



- **If *`break`* is used to skip the following *`switch`* statements, it has no effects on the outside loop**
  - `One-time use only`

```
18  /* loop until user types end-of-file key sequence */
19  while ( ( grade = getchar() ) != EOF ) {
20
21     /* determine which grade was input */
22     switch ( grade ) { /* switch nested in while */
23
24        case 'A': /* grade was uppercase A */
25        case 'a': /* or lowercase a */
26           ++aCount; /* increment aCount */
27           break; /* necessary to exit switch */
28
```