

02_03_C Structured Program Development

Objectives

In this chapter, you'll:

- Use basic problem-solving techniques.
- Develop algorithms through the process of top-down, stepwise refinement.
- Use the `if` selection statement and the `if...else` selection statement to select actions.
- Use the `while` iteration statement to execute statements in a program repeatedly.
- Use counter-controlled iteration and sentinel-controlled iteration.
- Learn structured programming.
- Use increment, decrement and assignment operators.

Why an ampersand in the **scanf** ?

- This is because parameters in C functions/procedures are passed by value, and the only way to pass by reference is to pass the reference (address, pointer to) the variable.
- In C, if we want to pass parameters by reference the address of (&) operator should be used.
- The **scanf()** function reads formatted input and has to put this input into something like a variable.
- What we are doing, in fact, is passing the address of such variable to the **scanf()** function so it can put the input (from the keyboard for example) directly in this variable.

```
1 // Fig. 3.6: fig03_06.c
2 // Class average program with counter-controlled iteration.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     unsigned int counter; // number of grade to be entered next
9     int grade; // grade value
10    int total; // sum of grades entered by user
11    int average; // average of grades
12
13    // initialization phase
14    total = 0; // initialize total
15    counter = 1; // initialize loop counter
16
17    // processing phase
18    while ( counter <= 10 ) { // loop 10 times
19        printf( "%s", "Enter grade: " ); // prompt for input
20        scanf( "%d", &grade ); // read grade from user
21        total = total + grade; // add grade to total
22        counter = counter + 1; // increment counter
23    } // end while
24
```

Fig. 3.6 | Class-average problem with counter-controlled iteration. (Part I of 2.)

```
25     // termination phase
26     average = total / 10; // integer division
27
28     printf( "Class average is %d\n", average ); // display result
29 } // end function main
```

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

Fig. 3.6 | Class-average problem with counter-controlled iteration. (Part 2 of 2.)

```
1 // Fig. 3.8: fig03_08.c
2 // Class-average program with sentinel-controlled iteration.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     unsigned int counter; // number of grades entered
9     int grade; // grade value
10    int total; // sum of grades
11
12    float average; // number with decimal point for average
13
14    // initialization phase
15    total = 0; // initialize total
16    counter = 0; // initialize loop counter
17
18    // processing phase
19    // get first grade from user
20    printf( "%s", "Enter grade, -1 to end: " ); // prompt for input
21    scanf( "%d", &grade ); // read grade from user
22
```

Fig. 3.8 | Class-average program with sentinel-controlled iteration. (Part I of 3.)

```
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( "%s", "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     puts( "No grades were entered" );
45 } // end else
46 } // end function main
```

Fig. 3.8 | Class-average program with sentinel-controlled iteration. (Part 2 of 3.)

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

```
Enter grade, -1 to end: -1
No grades were entered
```

Fig. 3.8 | Class-average program with sentinel-controlled iteration. (Part 3 of 3.)

```
1 // Fig. 3.10: fig03_10.c
2 // Analysis of examination results.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     // initialize variables in definitions
9     unsigned int passes = 0; // number of passes
10    unsigned int failures = 0; // number of failures
11    unsigned int student = 1; // student counter
12    int result; // one exam result
13
14    // process 10 students using counter-controlled loop
15    while ( student <= 10 ) {
16
17        // prompt user for input and obtain value from user
18        printf( "%s", "Enter result ( 1=pass,2=fail ): " );
19        scanf( "%d", &result );
20    }
```

Fig. 3.10 | Analysis of examination results. (Part I of 4.)

```
21     // if result 1, increment passes
22     if ( result == 1 ) {
23         passes = passes + 1;
24     } // end if
25     else { // otherwise, increment failures
26         failures = failures + 1;
27     } // end else
28
29     student = student + 1; // increment student counter
30 } // end while
31
32 // termination phase; display number of passes and failures
33 printf( "Passed %u\n", passes );
34 printf( "Failed %u\n", failures );
35
36 // if more than eight students passed, print "Bonus to instructor!"
37 if ( passes > 8 ) {
38     puts( "Bonus to instructor!" );
39 } // end if
40 } // end function main
```

Fig. 3.10 | Analysis of examination results. (Part 2 of 4.)

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Passed 6
Failed 4
```

Fig. 3.10 | Analysis of examination results. (Part 3 of 4.)

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Passed 9
Failed 1
Bonus to instructor!
```

Fig. 3.10 | Analysis of examination results. (Part 4 of 4.)

Assignment operator	Sample expression	Explanation	Assigns
<i>Assume: int c = 3, d = 5, e = 4, f = 6, g = 12;</i>			
+=	c += 7	c = c + 7	10 to c
-=	d -= 4	d = d - 4	1 to d
*=	e *= 5	e = e * 5	20 to e
/=	f /= 3	f = f / 3	2 to f
%=	g %= 9	g = g % 9	3 to g

Fig. 3.11 | Arithmetic assignment operators.

Operator	Sample expression	Explanation
++	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Fig. 3.12 | Increment and decrement operators

```
1 // Fig. 3.13: fig03_13.c
2 // Preincrementing and postincrementing.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     int c; // define variable
9
10    // demonstrate postincrement
11    c = 5; // assign 5 to c
12    printf( "%d\n", c ); // print 5
13    printf( "%d\n", c++ ); // print 5 then postincrement
14    printf( "%d\n\n", c ); // print 6
15
16    // demonstrate preincrement
17    c = 5; // assign 5 to c
18    printf( "%d\n", c ); // print 5
19    printf( "%d\n", ++c ); // preincrement then print 6
20    printf( "%d\n", c ); // print 6
21 }
```

Fig. 3.13 | Preincrementing and postincrementing. (Part I of 2.)

5
5
6

5
6
6

Fig. 3.13 | Preincrementing and postincrementing. (Part 2 of 2.)

Increment and Decrement Operators

- The three assignment statements in Fig. 3.10

```
passes = passes + 1;  
failures = failures + 1;  
student = student + 1;
```

can be written more concisely with *assignment operators* as

```
passes += 1;  
failures += 1;  
student += 1;
```

with *preincrement operators* as

```
++passes;  
++failures;  
++student;
```

or with *postincrement operators* as

```
passes++;  
failures++;  
student++;
```

Operators	Associativity	Type
<code>++</code> (<i>postfix</i>) <code>--</code> (<i>postfix</i>)	right to left	postfix
<code>+</code> <code>-</code> (<i>type</i>) <code>++</code> (<i>prefix</i>) <code>--</code> (<i>prefix</i>)	right to left	unary
<code>*</code> <code>/</code> <code>%</code>	left to right	multiplicative
<code>+</code> <code>-</code>	left to right	additive
<code><</code> <code><=</code> <code>></code> <code>>=</code>	left to right	relational
<code>==</code> <code>!=</code>	left to right	equality
<code>?:</code>	right to left	conditional
<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code>	right to left	assignment

Fig. 3.14 | Precedence and associativity of the operators encountered so far in the text.

```
1 // Fig. 2.5: fig02_05.c
2 // Addition program.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     int integer1; // first number to be entered by user
9     int integer2; // second number to be entered by user
10
11     printf( "Enter first integer\n" ); // prompt
12     scanf( "%d", &integer1 ); // read an integer
13
14     printf( "Enter second integer\n" ); // prompt
15     scanf( "%d", &integer2 ); // read an integer
16
17     int sum; // variable in which sum will be stored
18     sum = integer1 + integer2; // assign total to sum
19
20     printf( "Sum is %d\n", sum ); // print sum
21 }
```

Fig. 2.5 | Addition program. (Part 1 of 2.)