3

Structured Program Development in C

OBJECTIVES

In this chapter you will learn:

- To use nested `if...else` selection statements.
- The increment, decrement and assignment operators.

Outline

3.6 The `if…else` selection statement

- Nested `if…else` statements
  - Test for multiple cases by placing `if…else` selection statements inside `if…else` selection statement
  - Once condition is met, rest of statements skipped
  - Deep indentation usually not used in practice

3.6 The `if…else` selection statement

- Pseudocode for a nested `if...else` statement

```
If student's grade is greater than or equal to 90
  Print "A"
else
  If student's grade is greater than or equal to 80
     Print "B"
  else
     If student's grade is greater than or equal to 70
        Print "C"
     else
        If student's grade is greater than or equal to 60
           Print "D"
        else
           Print "F"
```

3.11 Assignment Operators

- Assignment operators abbreviate assignment expressions

```
c = c + 3;
```

can be abbreviated as `c += 3;` using the addition assignment operator

- Statements of the form

variable = *variable operator expression*;

can be rewritten as

variable `operator`= *expression*;

- Examples of other assignment operators:

```
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`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.

3.12 Increment and Decrement Operators

- **Increment operator (++)**
  - Can be used instead of `c+=1`
- **Decrement operator (--)**
  - Can be used instead of `c-=1`
- **Preincrement**
  - Operator is used before the variable (`++c` or `--c`)
  - Variable is changed before the expression it is in is evaluated
- **Postincrement**
  - Operator is used after the variable (`c++` or `c--`)
  - Expression executes before the variable is changed

3.12 Increment and Decrement Operators

- If `c` equals 5, then

  ```
  printf( "%d", ++c );
  ```

  - Prints 6

  ```
  printf( "%d", c++ );
  ```

  - Prints 5
  - In either case, `c` now has the value of 6
- When variable not in an expression
  - Preincrementing and postincrementing have the same effect

  ```
  ++c;
  printf( “%d”, c );
  ```

  - Has the same effect as

  ```
  c++;
  printf( “%d”, c );
  ```

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.

Outline

fig03_13.c

```c
/* Fig. 3.13: fig03_13.c
   Preincrementing and postincrementing */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int c;                   /* define variable */

   /* demonstrate postincrement */
   c = 5;                   /* assign 5 to c */
   printf( "%d\n", c );     /* print 5 */
   printf( "%d\n", c++ ); /* print 5 then postincrement */
   printf( "%d\n\n", c ); /* print 6 */

   /* demonstrate preincrement */
   c = 5;                   /* assign 5 to c */
   printf( "%d\n", c );     /* print 5 */
   printf( "%d\n", ++c ); /* preincrement then print 6 */
   printf( "%d\n", c );     /* print 6 */

   return 0; /* indicate program ended successfully */

} /* end function main */
```

```
5
5
6

5
6
6
```

`c` is printed, then incremented

`c` is incremented, then printed

Good Programming Practice 3.7

Unary operators should be placed directly next to their operands with no intervening spaces.

Common Programming Error 3.10

Attempting to use the increment or decrement operator on an expression other than a simple variable name is a syntax error, e.g., writing ++ `(x + 1)`.

Error-Prevention Tip 3.4

C generally does not specify the order in which an operator's operands will be evaluated (although we will see exceptions to this for a few operators in Chapter 4). Therefore you should avoid using statements with increment or decrement operators in which a particular variable being incremented or decremented appears more than once.

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

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