BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

Programming and Computer Applications-1

Control Statements

Instructor : PhD, Associate Professor Leyla Muradkhanli

Outline

- Control Structures
- if selection statement
- if … else selection statement
- while iteration statement
- Assignment Operators
- Increment and Decrement Operators

Control Structures

- C provides three types of selection structures in the form of statements.
- The `if` selection statement either performs (selects) an action if a condition is true or skips the action if the condition is false.
- The `if`…`else` selection statement performs an action if a condition is true and performs a different action if the condition is false.
- The `switch` selection statement performs one of many different actions depending on the value of an expression.

Control Structures

- The `if` statement is called a single-selection statement because it selects or ignores a single action.
- The `if`…`else` statement is called a double-selection statement because it selects between two different actions.
- The `switch` statement is called a multiple-selection statement because it selects among many different actions.
- C provides three types of repetition structures in the form of statements, namely `while`, `do`…`while`, and `for`.

Control Structures

- C has only seven control statements: sequence, three types of selection and three types of repetition.
- Each C program is formed by combining as many of each type of control statement as is appropriate for the algorithm the program implements.
- As with the sequence structure of Fig. 3.1, we’ll see that the flowchart representation of each control statement has two small circle symbols, one at the entry point to the control statement and one at the exit point.
- These single-entry/single-exit control statements make it easy to build programs.

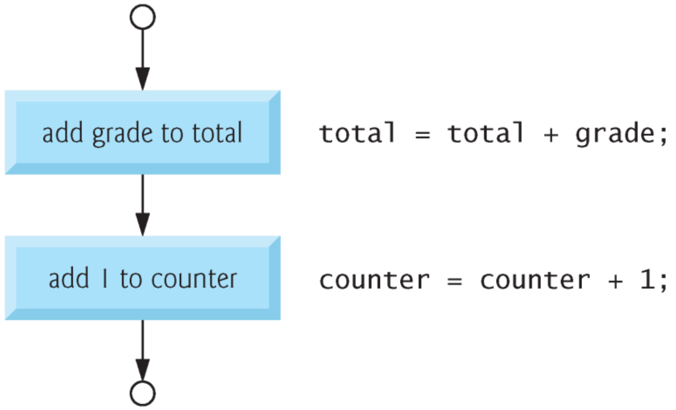

Fig. 3.1 | Flowcharting C's sequence structure.

Control Structures

- The control-statement flowchart segments can be attached to one another by connecting the exit point of one control statement to the entry point of the next.
- This is much like the way in which a child stacks building blocks, so we call this control-statement stacking.
- We’ll learn that there is only one other way control statements may be connected—a method called control-statement nesting.
- Thus, any C program we’ll ever need to build can be constructed from only seven different types of control statements combined in only two ways.
- This is the essence of simplicity.

The if Selection Statement

- Selection structures are used to choose among alternative courses of action.
- For example, suppose the passing grade on an exam is 60.
- The pseudocode statement

  -
    ```
    If student’s grade is greater than or equal to 60
    Print “Passed”
    ```
- determines if the condition “student’s grade is greater than or equal to 60” is true or false.
- If the condition is true, then “Passed” is printed, and the next pseudocode statement in order is “performed” (remember that pseudocode is not a real programming language).

The if Selection Statement

- If the condition is false, the printing is ignored, and the next pseudocode statement in order is performed.
- The second line of this selection structure is indented.
- Such indentation is optional, but it’s highly recommended as it helps emphasize the inherent structure of structured programs.
- The C compiler ignores white-space characters like blanks, tabs and newlines used for indentation and vertical spacing.

The if Selection Statement

- The preceding pseudocode *If statement may be written in C as*

  -
    ```c
    if ( grade >= 60 ) {
       printf( "Passed\n" );
    } /* end if */
    ```
- Notice that the C code corresponds closely to the pseudocode.

The if Selection Statement

- The flowchart of Fig. 3.2 illustrates the single-selection `if` statement.
- This flowchart contains what is perhaps the most important flowcharting symbol—the diamond symbol, also called the decision symbol, which indicates that a decision is to be made.
- The decision symbol contains an expression, such as a condition, that can be either true or false.

The if Selection Statement

- The decision symbol has two flowlines emerging from it.
- One indicates the direction to take when the expression in the symbol is true; the other indicates the direction to take when the expression is false.
- Decisions can be based on conditions containing relational or equality operators.
- In fact, a decision can be based on any expression—if the expression evaluates to zero, it’s treated as false, and if it evaluates to nonzero, it’s treated as true.

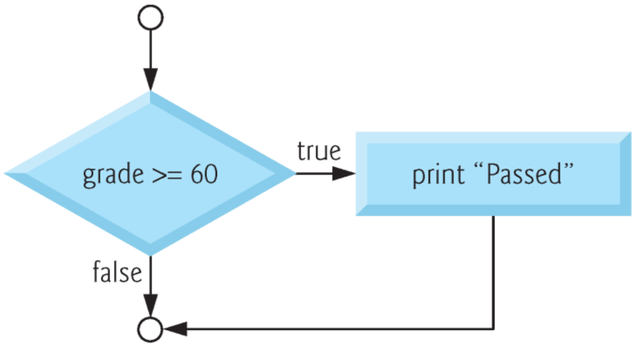

Fig. 3.2 | Flowcharting the single-selection `if` statement.

The if…else Selection Statement

- The `if…else` selection statement allows you to specify that different actions are to be performed when the condition is true than when the condition is false.
- For example, the pseudocode statement

  -
    ```
    If student’s grade is greater than or equal to 60
    Print “Passed”
    else
    Print “Failed”
    ```
- prints *Passed if the student’s grade is greater than or equal to 60 and prints Failed if the student’s grade is less than 60.*
- In either case, after printing occurs, the next pseudocode statement in sequence is “performed.” The body of the *else is also indented.*

The if…else Selection Statement

- The preceding pseudocode *If…else statement may be written in C as*

  -
    ```
    if ( grade >= 60 ) {
       printf( "Passed\n" );
    } /* end if */
    else {
       printf( "Failed\n" );
    } /* end else */
    ```
- The flowchart of Fig. 3.3 nicely illustrates the flow of control in the `if`…`else` statement.
- Once again, note that (besides small circles and arrows) the only symbols in the flowchart are rectangles (for actions) and a diamond (for a decision).

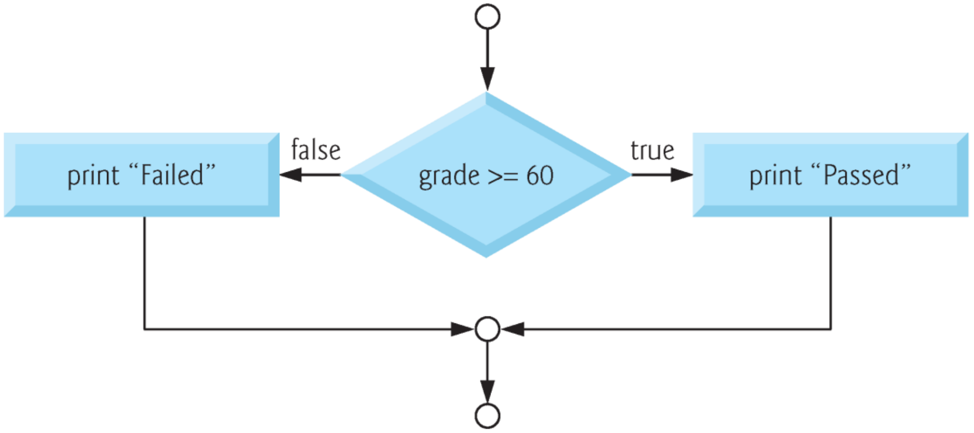

Fig. 3.3 | Flowcharting the double-selection `if...else` statement.

The if…else Selection Statement

- C provides the conditional operator (`?:`) which is closely related to the `if`…`else` statement.
- The conditional operator is C’s only ternary operator—it takes three operands.
- The operands together with the conditional operator form a conditional expression.
- The first operand is a condition.
- The second operand is the value for the entire conditional expression if the condition is true and the third operand is the value for the entire conditional expression if the condition is false.

The if…else Selection Statement

- For example, the `printf` statement
    - `printf( "%s\n", grade >= 60 ? "Passed" : "Failed" );`
- contains a conditional expression that evaluates to the string literal `"Passed"` if the condition `grade >= 60` is true and evaluates to the string literal `"Failed"` if the condition is false.
- The format control string of the `printf` contains the conversion specification `%s` for printing a character string.
- So the preceding `printf` statement performs in essentially the same way as the preceding `if`…`else` statement.

The if…else Selection Statement

- The second and third operands in a conditional expression can also be actions to be executed.
- For example, the conditional expression

    -
      ```
      grade >= 60 ? printf( "Passed\n" ) :
          printf( "Failed\n" );
      ```
- is read, “If `grade` is greater than or equal to `60` then `printf("Passed\n")`, otherwise `printf( "Failed\n" )`.” This, too, is comparable to the preceding `if`…`else` statement.
- We’ll see that conditional operators can be used in some situations where `if`…`else` statements cannot.

The if…else Selection Statement

- Nested if…else statements test for multiple cases by placing if…else statements inside if…else statements.
- For example, the following pseudocode statement will print A for exam grades greater than or equal to 90, B for grades greater than or equal to 80, C for grades greater than or equal to 70, D for grades greater than or equal to 60, and F for all other grades.

  -
    ```
    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”
    ```

The if…else Selection Statement

- This pseudocode may be written in C as

```
if ( grade >= 90 )
   printf( "A\n" );
else
   if ( grade >= 80 )
      printf("B\n");
   else
      if ( grade >= 70 )
         printf("C\n");
      else
         if ( grade >= 60 )
            printf( "D\n" );
         else
            printf( "F\n" );
```

The if…else Selection Statement

- If the variable `grade` is greater than or equal to 90, the first four conditions will be true, but only the `printf` statement after the first test will be executed.
- After that `printf` is executed, the `else` part of the “outer” `if`…`else` statement is skipped.

The if…else Selection Statement

- Many C programmers prefer to write the preceding `if` statement as

  -
    ```
    if ( grade >= 90 )
       printf( "A\n" );
    else if ( grade >= 80 )
       printf( "B\n" );
    else if ( grade >= 70 )
       printf( "C\n" );
    else if ( grade >= 60 )
       printf( "D\n" );
    else
       printf( "F\n" );
    ```

The if…else Selection Statement

- As far as the C compiler is concerned, both forms are equivalent.
- The latter form is popular because it avoids the deep indentation of the code to the right.
- The `if` selection statement expects only one statement in its body.
- To include several statements in the body of an `if`, enclose the set of statements in braces (`{` and `}`).
- A set of statements contained within a pair of braces is called a compound statement or a block.

The if…else Selection Statement

- The following example includes a compound statement in the `else` part of an `if…else` statement.

  -
    ```
    if ( grade >= 60 ) {
       printf( "Passed.\n" );
    } /* end if */
    else {
       printf( "Failed.\n" );
       printf( "You must take this course again.\n" );
    } /* end else */
    ```

The if…else Selection Statement

- In this case, if grade is less than `60`, the program executes both `printf` statements in the body of the `else` and prints

  -
    ```
    Failed.
    You must take this course again.
    ```
- Notice the braces surrounding the two statements in the `else` clause.
- These braces are important. `Without the braces, the statement`

  ```
  printf( "You must take this course again.\n" );
  ```
- would be outside the body of the `else` part of the `if`, and would execute regardless of whether the grade was less than 60.

The if…else Selection Statement

- A syntax error is caught by the compiler.
- A logic error has its effect at execution time.
- A fatal logic error causes a program to fail and terminate prematurely.
- A nonfatal logic error allows a program to continue executing but to produce incorrect results.

The while Repetition Statement

- A repetition statement allows you to specify that an action is to be repeated while some condition remains true.
- The pseudocode statement
  - While there are more items on my shopping list
    Purchase next item and cross it off my list
- describes the repetition that occurs during a shopping trip.
- The condition, “there are more items on my shopping list” may be true or false.
- If it’s true, then the action, “Purchase next item and cross it off my list” is performed.
- This action will be performed repeatedly while the condition remains true.

The while Repetition Statement

- The statement(s) contained in the *while repetition statement constitute the body of the while.*
- The *while statement body may be a single statement or a compound statement.*
- Eventually, the condition will become false (when the last item on the shopping list has been purchased and crossed off the list).
- At this point, the repetition terminates, and the first pseudocode statement after the repetition structure is executed.

The while Repetition Statement

- As an example of an actual `while`, consider a program segment designed to find the first power of 3 larger than 100.
- Suppose the integer variable `product` has been initialized to 3.
- When the following `while` repetition statement finishes executing, `product` will contain the desired answer:
  - `product = 3;`

  -
    ```
    while ( product <= 100 ) {
        product = 3 * product;
    } /* end while */
    ```
- The flowchart of Fig. 3.4 nicely illustrates the flow of control in the `while` repetition statement.

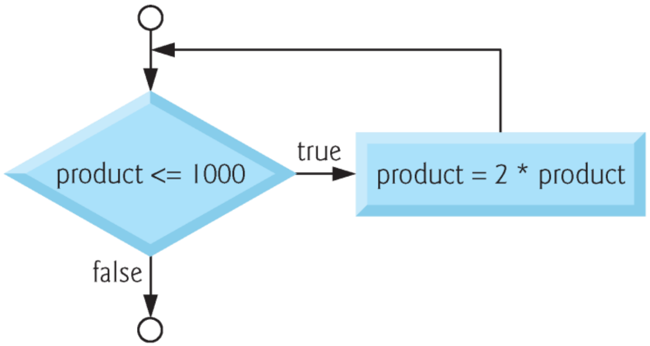

Fig. 3.4 | Flowcharting the `while` repetition statement.

The while Repetition Statement

- When the `while` statement is entered, the value of `product` is 3.
- The variable `product` is repeatedly multiplied by 3, taking on the values 9, 27 and 81 successively.
- When `product` becomes 243, the condition in the `while` statement, `product <= 100`, becomes false.
- This terminates the repetition, and the final value of `product` is 243.
- Program execution continues with the next statement after the `while`.

Assignment Operators

- C provides several assignment operators for abbreviating assignment expressions.
- For example, the statement
    - `c = c + 3;`
- can be abbreviated with the addition assignment operator += as
    - `c += 3;`
- The **+=** operator adds the value of the expression on the right of the operator to the value of the variable on the left of the operator and stores the result in the variable on the left of the operator.

Assignment Operators

- Any statement of the form
    - *`variable = variable operator expression;`*
- where *operator is one of the binary operators +, -, *, / or %, can be written in the form*
    - *`variable operator= expression;`*
- Thus the assignment `c += 3` adds `3` to `c`.
- Figure 3.11 shows the arithmetic assignment operators, sample expressions using these operators and explanations.

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.

Increment and Decrement Operators

- C also provides the unary increment operator, ++, and the unary decrement operator, −−, which are summarized in Fig. 3.12.
- If a variable `c` is incremented by 1, the increment operator `++` can be used rather than the expressions `c = c + 1` or `c += 1`.
- If increment or decrement operators are placed before a variable (i.e., prefixed), they’re referred to as the preincrement or predecrement operators, respectively.
- If increment or decrement operators are placed after a variable (i.e., postfixed), they’re referred to as the postincrement or postdecrement operators, respectively.

Increment and Decrement Operators

- Preincrementing (predecrementing) a variable causes the variable to be incremented (decremented) by 1, then the new value of the variable is used in the expression in which it appears.
- Postincrementing (postdecrementing) the variable causes the current value of the variable to be used in the expression in which it appears, then the variable value is incremented (decremented) by 1.

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

Increment and Decrement Operators

- Figure 3.13 demonstrates the difference between the preincrementing and the postincrementing versions of the `++` operator.
- Postincrementing the variable `c` causes it to be incremented after it’s used in the `printf` statement.
- Preincrementing the variable `c` causes it to be incremented before it’s used in the `printf` statement.

```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 */
```

Fig. 3.13 | Preincrementing vs. postincrementing. (Part 1 of 2.)

```
5
5
6
  s
5
6
6
```

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

Increment and Decrement Operators

- The program displays the value of `c` before and after the `++` operator is used.
- The decrement operator (`--`) works similarly.

Increment and Decrement Operators

- The three assignment statements

  -
    ```
    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++;
    ```

Increment and Decrement Operators

- It’s important to note here that when incrementing or decrementing a variable in a statement by itself, the preincrement and postincrement forms have the same effect.

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 and associativity of the operators encountered so far in the text.