Chapter 3 part 1 Structured Program Development in C

C How to Program

Introduction

- Before writing a program to solve a particular problem, we must have a thorough understanding of the problem and a carefully planned solution approach.
- The next two chapters discuss techniques that facilitate the development of structured computer programs.

Algorithms

- The solution to any computing problem involves executing a series of actions in a specific order.
- A procedure for solving a problem in terms of
  - the actions to be executed, and
  - the order in which these actions are to be executed
- is called an algorithm.
- Correctly specifying the order in which the actions are to be executed is important.

Algorithms (Cont.)

- Consider the “rise-and-shine algorithm” followed by one junior executive for getting out of bed and going to work: (1) Get out of bed, (2) take off pajamas, (3) take a shower, (4) get dressed, (5) eat breakfast, (6) carpool to work.
- This routine gets the executive to work well prepared to make critical decisions.

Algorithms (Cont.)

- Suppose that the same steps are performed in a slightly different order: (1) Get out of bed, (2) take off pajamas, (3) get dressed, (4) take a shower, (5) eat breakfast, (6) carpool to work.
- In this case, our junior executive shows up for work soaking wet.
- Specifying the order in which statements are to be executed in a computer program is called program control.

Pseudocode

- Pseudocode is an artificial and informal language that helps you develop algorithms.
- Pseudocode is similar to everyday English; it’s convenient and user friendly although it’s not an actual computer programming language.
- Pseudocode programs are *not* executed on computers.
- Rather, they merely help you “think out” a program before attempting to write it in a programming language like C.

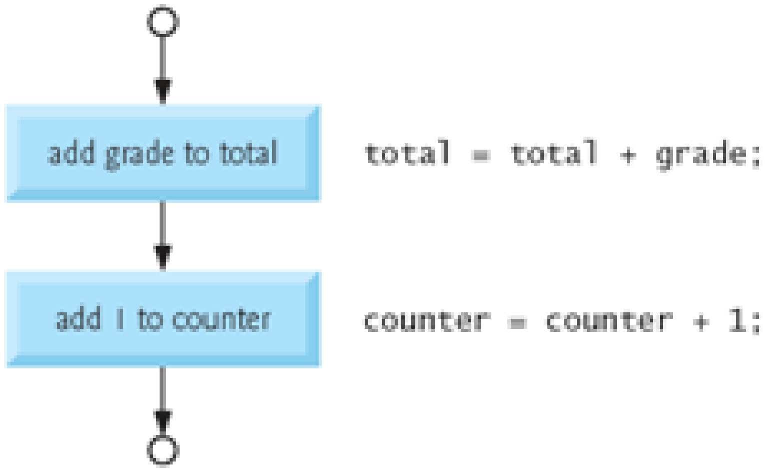
Pseudocode (Cont.)

- A carefully prepared pseudocode program may be converted easily to a corresponding C program.
- Pseudocode consists only of action statements—those that are executed when the program has been converted from pseudocode to C and is run in C.

- *If studet's grade is greater than 60*
  - *Print "Passed"*

Flowcharts

- A flowchart is a graphical representation of an algorithm or of a portion of an algorithm.
- Flowcharts are drawn using certain special-purpose symbols such as rectangles, diamonds, rounded rectangles, and small circles; these symbols are connected by arrows called flowlines.



Control Structures

- Normally, statements in a program are executed one after the other in the order in which they’re written.
- This is called sequential execution.

- Various C statements we’ll soon discuss enable you to specify that the next statement to be executed may be other than the next one in sequence.
- This is called transfer of control.

Control Structures (Cont.)

- Research had demonstrated that all programs could be written in terms of only three control structures, namely the sequence structure, the selection structure and the repetition structure.
- The sequence structure is simple—unless directed otherwise, the computer executes C statements one after the other in the order in which they're written.

Control Structures (Cont.)

Selection Statements in C

- C provides three types of selection structures in the form of statements.

- 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.

Control Structures (Cont.)

Repetition Statements in C

- C provides three types of repetition structures in the form of statements, namely:
- `while`
- `do…while` (discussed in Chapter 4).
- `for` (discussed in Chapter 4).

The `if` Selection Statement

- Selection statements 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 whether the condition "student's grade is greater than or equal to 60" is true or false.

The `if` Selection Statement (Cont.)

```
If student's grade is greater than or equal to 60
    Print "Passed"
```

- If the condition is true, then "Passed" is printed, and the next pseudocode statement in order is performed
- If the condition is false, the printing is ignored, and the next pseudocode statement in order is performed.

The `if` Selection Statement (Cont.)

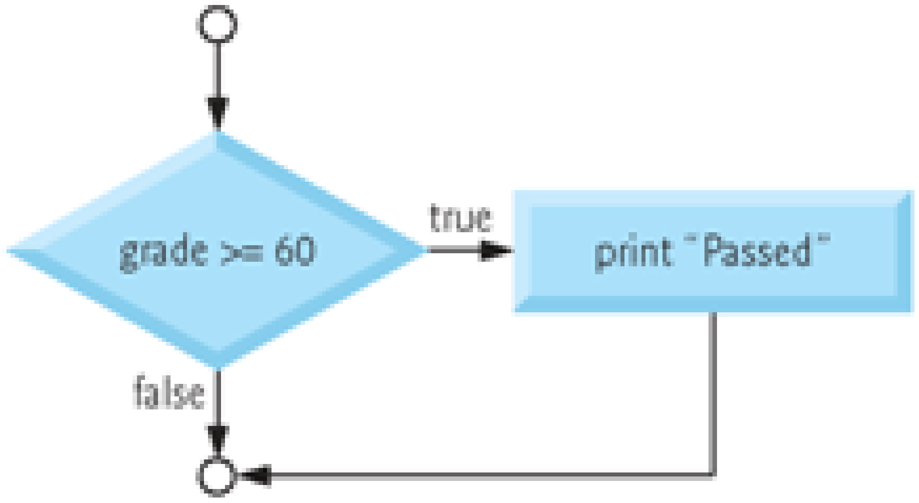
- The preceding pseudocode *If statement may be written in C as*

  -
    ```
    if ( grade >= 60 ) {
       printf( "Passed\n" );
    } /* end if */
    ```

- Notice that the C code corresponds closely to the pseudocode (of course you'll also need to declare the `int` variable `grade`).

The `if` Selection Statement (Cont.)

- The decision symbol contains an expression, such as a condition, that can be either true or false. The flowchart illustrates the flow of control in the `if` statement



The `if` Selection Statement (Cont.)

- 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.

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 and when it's 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 *Failed* if the student's grade is less than 60.

The `if...else` Selection Statement (Cont.)

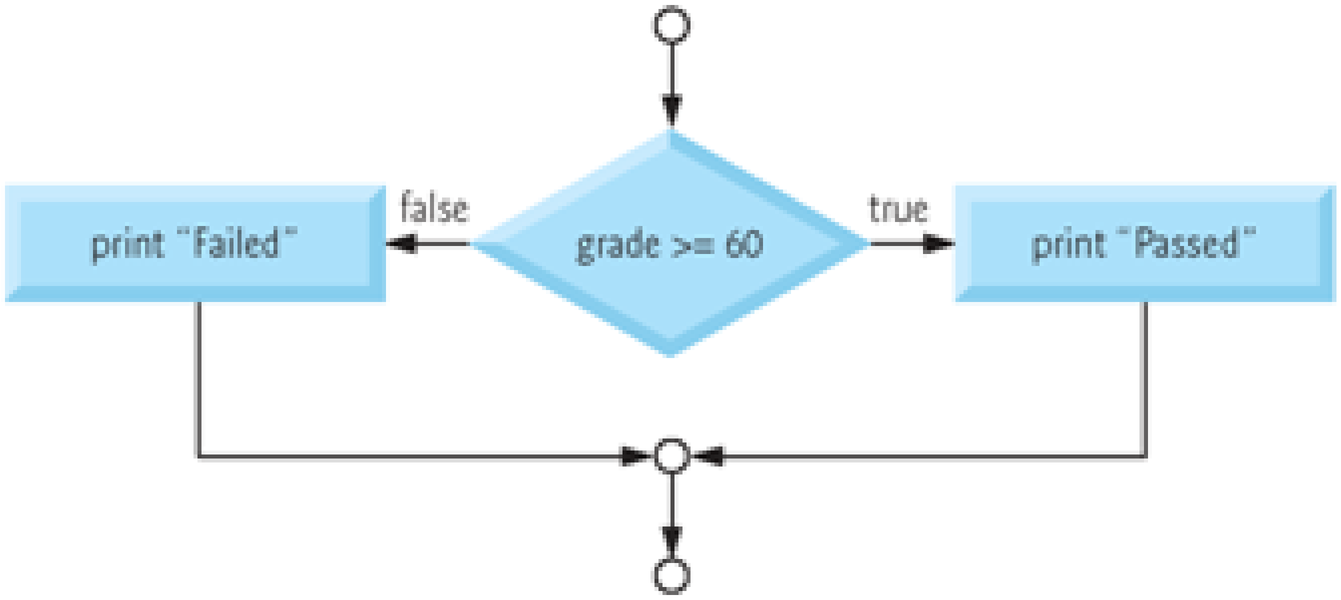
- The preceding pseudocode *If…else* statement may be written in C as

  -
    ```
    if ( grade >= 60 ) {
       printf( "Passed\n" );
    } /* end if */
    else {
       printf( "Failed\n" );
    } /* end else */
    ```

- *Create a program in Codeblocks that uses the code above*

The if...else Selection Statement (Cont.)

- The flowchart illustrates the flow of control in the if…else statement.



Conditional Operator (?:)

```
printf( grade >= 60 ? "Passed" : "Failed" );
```

- (first ? Second : third);
- The first operand is a *condition*.
- The second operand is the value for the entire conditional expression if the condition is *true*
- The third operand is the value for the entire conditional expression if the condition is *false*

Conditional Operator (?:)

```
printf( grade >= 60 ? "Passed" : "Failed" );
```

- The conditional operator (`?:`) is closely related to the `if`…`else` statement.
- The conditional operator is C’s only ternary operator—it takes *three* operands.

Conditional Operator (?:)

- For example, the following statement
  - printf(grade >= **60** ? **"Passed"** : **"Failed"**);

Contains a conditional expression that evaluates to the string *"Passed"* if the condition *grade >= 60* is true and to the string *"Failed"* if the condition is false.

The `if…else` Selection Statement (Cont.)

Nested if...else Statements

- Nested `if…else` statements test for multiple cases by placing `if…else` statements inside `if…else` statements.

The `if...else` Selection Statement (Cont.)

- The `if` selection statement expects only one statement in its body—if you have only one statement in the if's body, you do not need to enclose it in braces.
- To include several statements in the body of an `if`, you must 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 (Cont.)

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

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

- *Create a program in Codeblocks that uses the code above*

The `if...else` Selection Statement (Cont.)

- 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.
    ```

- The braces surrounding the two statements in the `else` are important. Without them, the statement

```
printf( "You must take this course again." );
```

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 (Cont.)

- Just as a compound statement (block) can be placed anywhere a single statement can be placed, it’s also possible to have no statement at all, i.e., the empty statement.
    - The empty statement is represented by placing a semicolon (;) where a statement would normally be.

The `if...else` Selection Statement (Cont.)

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

- *Create a program in Codeblocks that uses the code above*

The `if...else` Selection Statement (Cont.)

- If the variable *`grade`* is greater than or equal to 90, all 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

- You may prefer to write the preceding `if` statement as

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

- *Create a program in Codeblocks that uses the code above*

Self-Review exercise

- Write a program that reads an integer and determines if it is odd or even.
- Define variable x of type int.
- Use the if statement to determine if the number is even or odd.
- An even number is a multiple of two, and any multiple of two leaves a remainder of zero when divided by 2.

The `while` Repetition Statement

- A repetition statement (also called an iteration 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 `while` Repetition Statement

- *While there are more items on my shopping list*
  Purchase next item and cross it off my list

- 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 (Cont.)

- The statement(s) contained in the *while* repetition statement constitute the body of the while.
- 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 statement *after* the repetition structure is executed.

The while Repetition Statement (Cont.)

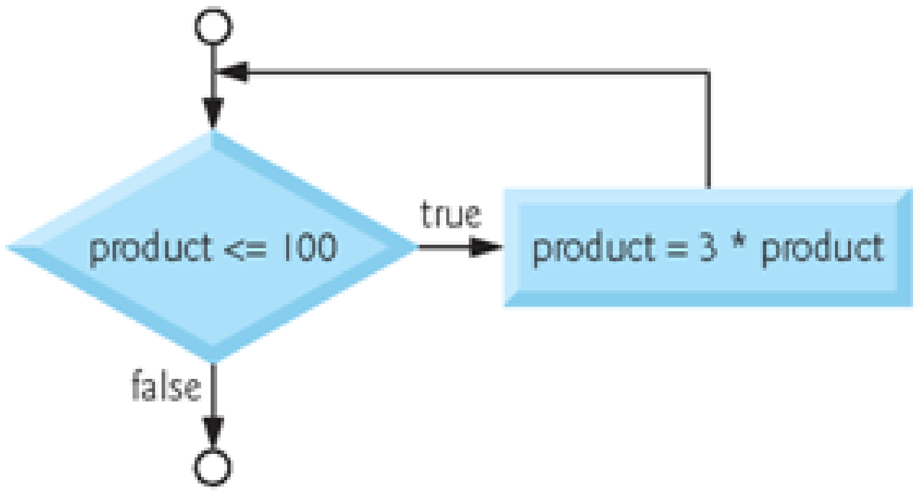
- As an example of a *while* statement, consider a program segment designed to find the first power of 3 larger than 100.

- 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 while Repetition Statement (Cont.)

```
product = 3;
while ( product <= 100 ) {
   product = 3 * product;
} /* end while */
```



The while Repetition Statement (Cont.)

```
product = 3;
while ( product <= 100 ) {
   product = 3 * product;
} /* end while */
```

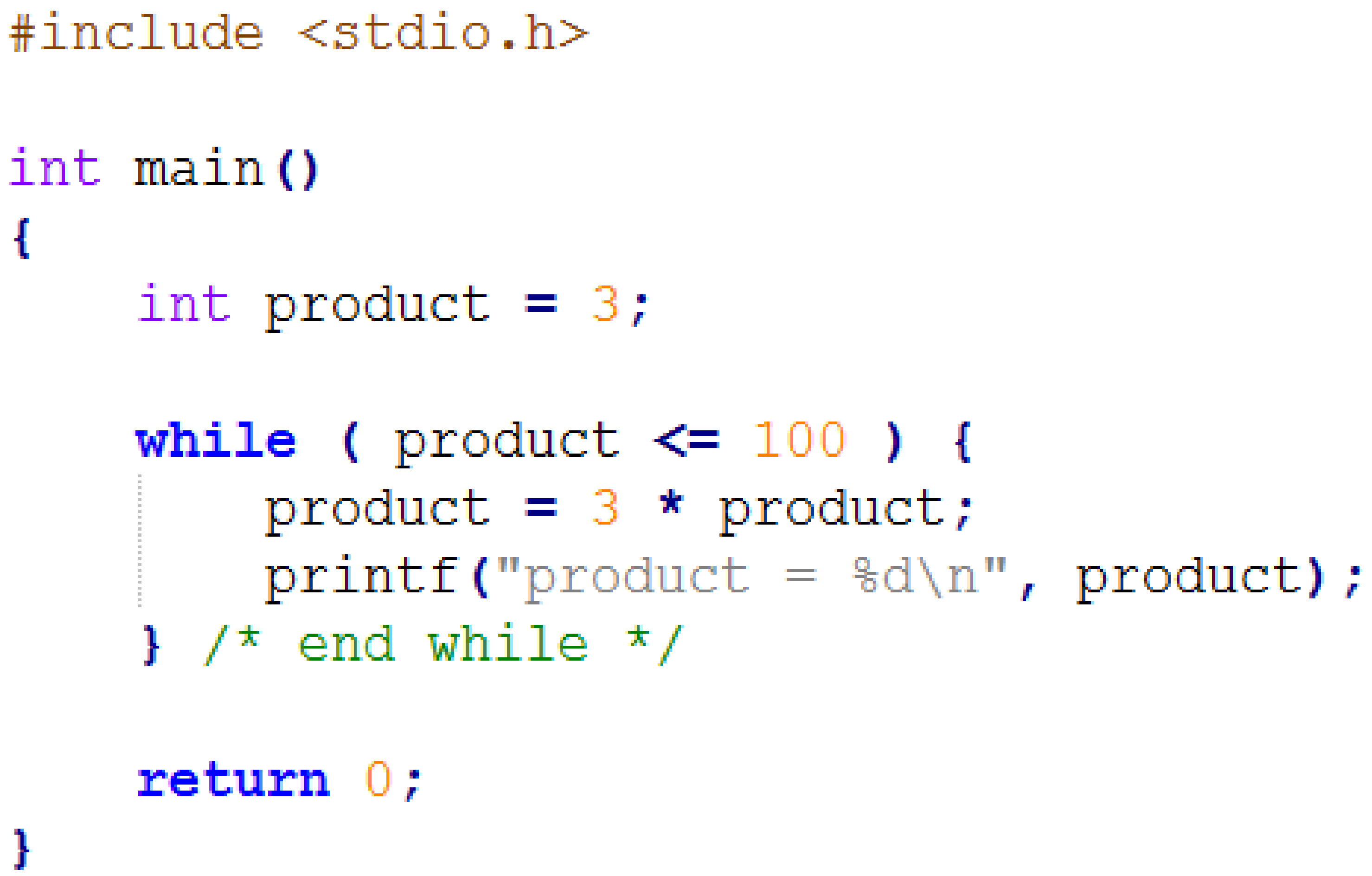
- When the `while` statement is entered, the value of `product` is 3.
- The variable `product` is repeatedly multiplied by 3, taking on the values 9, 27 and 81 successively.

The while Repetition Statement (Cont.)

```
product = 3;
while ( product <= 100 ) {
   product = 3 * product;
} /* end while */
```

- 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`.

Type this program in Codeblocks



```c
#include <stdio.h>

int main()
{
    int product = 3;

    while ( product <= 100 ) {
        product = 3 * product;
        printf("product = %d\n", product);
    } /* end while */

    return 0;
}
```

Self-Review exercise

- Write a program that calculates the sum of the integers from 1 to 10.
- Define variables sum and x of type int.
- Use the while statement to loop through the calculation and increment statements.
- The loop should terminate when the value of x becomes 11.