C++ Programming:
Control Structures

Some material taken from: C++ Programming: Program Design Including Data Structures

Objectives

- Learn about control structures
- Examine relational and logical operators
- Explore how to form and evaluate logical (Boolean) expressions
- Discover how to use the selection control structures if, and if...else in a program
- Learn about the repetition (looping) control structures while and for

Control Structures

- A computer can proceed:
  - In sequence
  - Selectively (branch) - making a choice
  - Repetitively (iteratively) - looping
- Some statements are executed only if certain conditions are met
- A condition is represented by a logical (Boolean) expression that can be true or false
- A condition is met if it evaluates to true

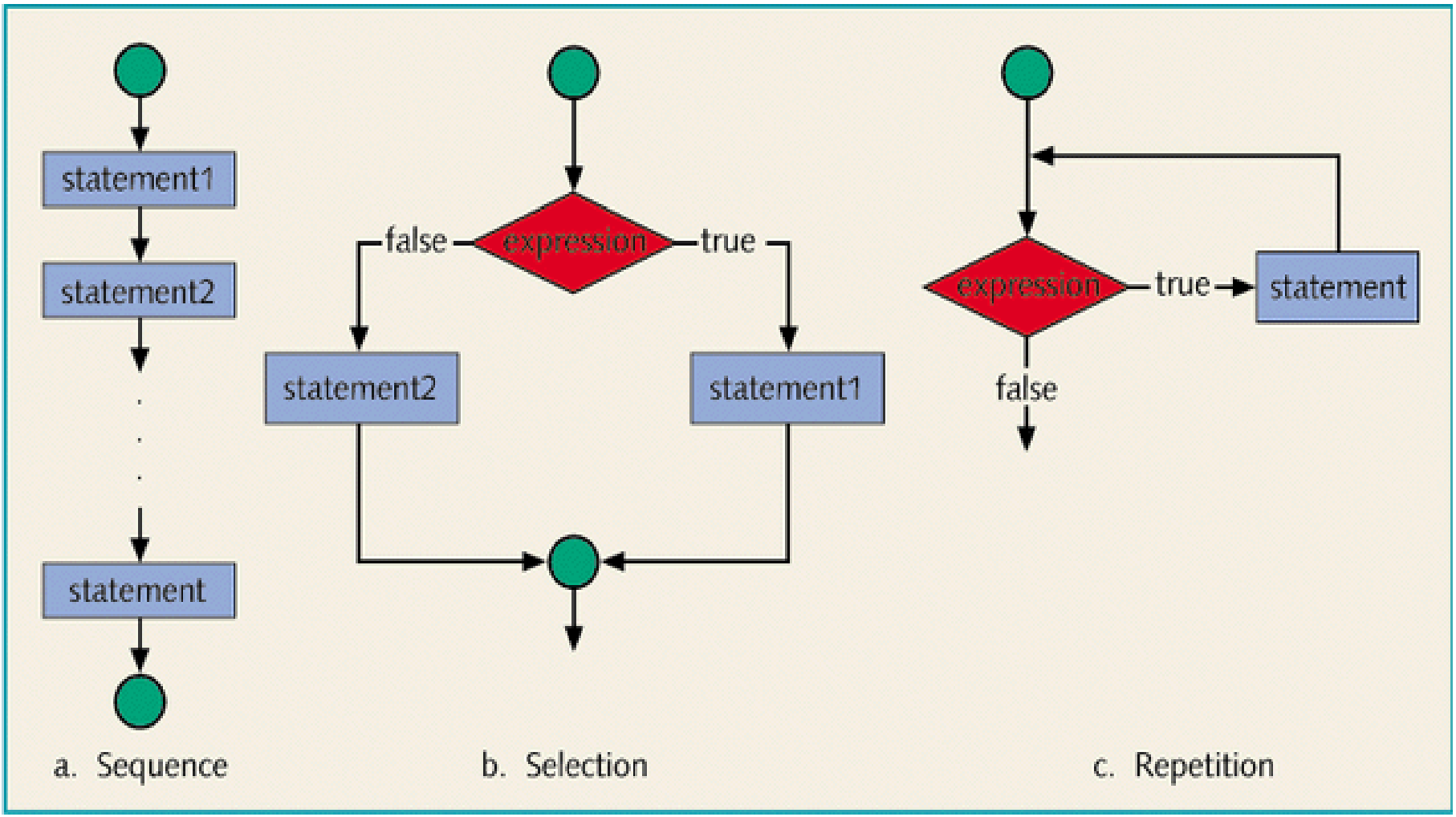

Figure 4-1 Flow of execution

Relational Operators

- Relational operators:
  - Allow comparisons
  - Require two operands (binary)
  - Return 1 if expression is true, 0 otherwise
- Comparing values of different data types may produce unpredictable results
  - For example, 8 < '5' should not be done
- Any nonzero value is treated as true

Table 4-1 Relational Operators in C++

Operator	Description
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Logical (Boolean) Operators

- Logical (Boolean) operators enable you to combine logical expressions
- Three logical (Boolean) operators:
  - ! - not
  - && – and
  - || - or
- Logical operators take logical values as operands and yield logical values as results
- ! is unary; && and || are binary operators
- Putting ! in front of a logical expression reverses its value

Table 4-5 The ! (not) Operator

Expression	!(Expression)
`true` (nonzero)	`false` (0)
`false` (0)	`true` (1)

Table 4-6 The && (and) Operator

Expression1	Expression2	Expression1 && Expression2
`true` (nonzero)	`true` (nonzero)	`true` (1)
`true` (nonzero)	`false` (0)	`false` (0)
`false` (0)	`true` (nonzero)	`false` (0)
`false` (0)	`false` (0)	`false` (0)

Table 4-7 The || (or) Operator

Expression1	Expression2	Expression1 || Expression2
`true` (nonzero)	`true` (nonzero)	`true` (1)
`true` (nonzero)	`false` (0)	`true` (1)
`false` (0)	`true` (nonzero)	`true` (1)
`false` (0)	`false` (0)	`false` (0)

Order of precedence of operators

Table 4-8 Precedence of Operators

Operators	Precedence
!, +, - (unary operators)	first
*, /, %	second
+, -	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

- Same precedence operators are evaluated from left to right
- Use parenthesis for clarity

Caution with mixing binary operators

- The following expression appears to represent a comparison of 0, num, and 10:

  0 <= num <= 10

- It always evaluates true because 0 <= num evaluates to either 0 or 1, and 0 <= 10 is true and 1 <= 10 is true
- The correct way to write this expression is:

  0 <= num && num <= 10

Short-Circuit Evaluation

- Short-circuit evaluation: evaluation of a logical expression in C++ starts from left to right and stops (for efficiency) as soon as the value of the expression is known
- Example:

```
(age >= 21) || ( x == 5)              //Line 1

(grade == 'A') && (x >= 7)            //Line 2
```

Logical (Boolean) Expressions

- The bool Data Type and Logical (Boolean) Expressions
  - The data type bool has logical (Boolean) values true and false
  - bool, true, and false are reserved words
  - The identifier true has the value 1
  - The identifier false has the value 0

Example

- Logical operators
- LogicalOperators.cpp

Selection

One-Way (if) Selection

- The syntax of one-way selection is:

  if(expression)

  statement

- Statement is executed if the value of the expression is true
- Statement is bypassed if the value is false; program goes to the next statement

One-Way (if) Selection (continued)

- The expression is sometimes called a decision maker because it decides whether to execute the statement that follows it
- The statement following the expression is sometimes called the action statement
- The expression is a logical expression
- The statement is any C++ statement
- if is a reserved word

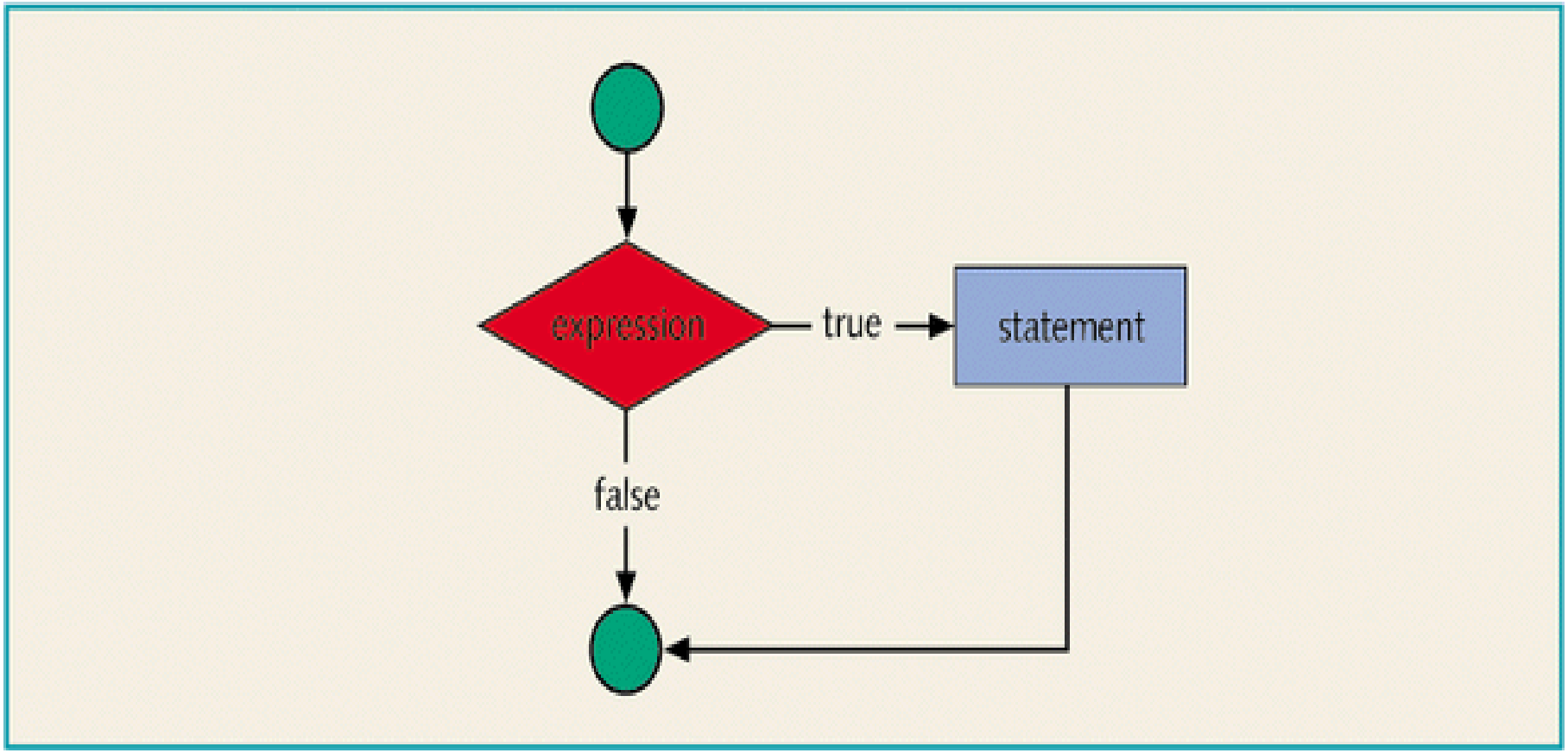

Figure 4-2 One-way selection

Two-Way (if…else) Selection

- Two-way selection takes the form:

```
if(expression)
        statement1
else
        statement2
```

- If expression is true, statement1 is executed otherwise statement2 is executed
- statement1 and statement2 are any C++ statements
- else is a reserved word

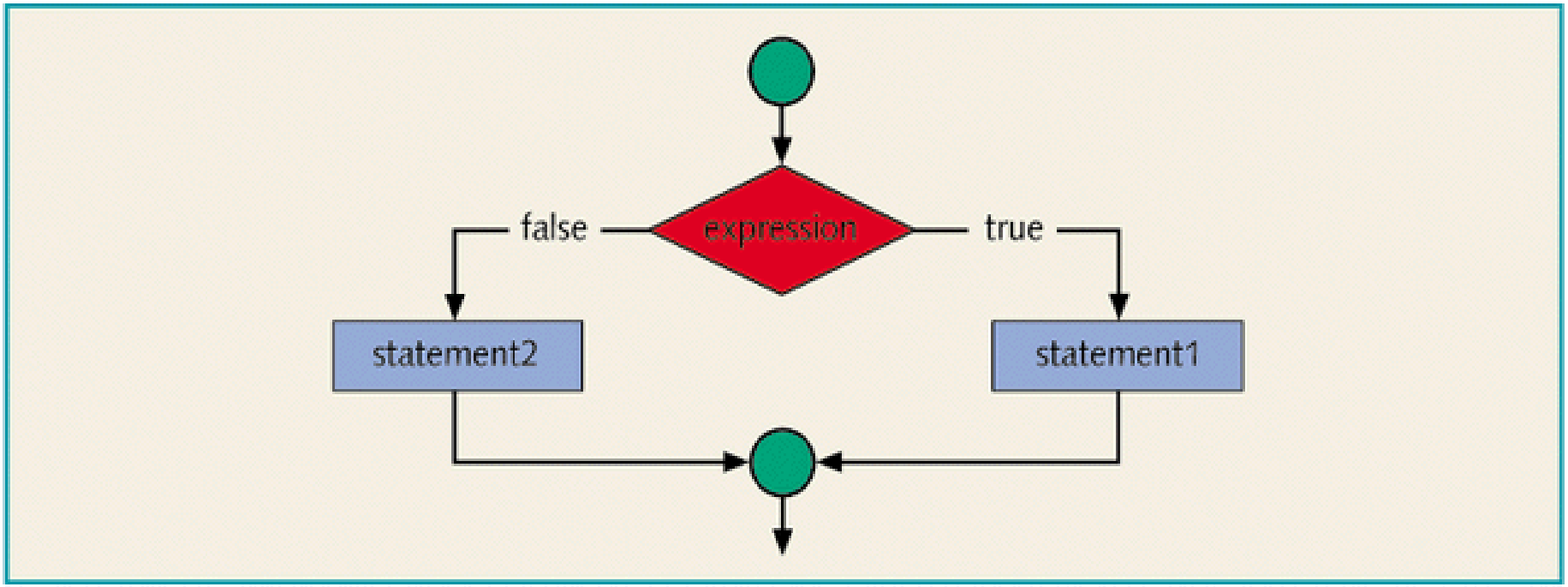

Figure 4-3 Two-way selection

Compound (Block of) Statement

- Compound statement (block of statements):

```
{
    statement1;
    statement2;
    .
    .
    .
    statementn;
}
```

- A compound statement is a single statement

Compound Statement Example

```
if(age > 21)
{
    cout<<" Eligible to vote."<<endl;
    cout<<" No longer a minor."<<endl;
}
else
{
    cout<<"Not eligible to vote."<<endl;
    cout<<"Still a minor."<<endl;
}
```

Nested if

- Nesting: one control statement in another
- An else is associated with the most recent if that has not been paired with an else
- For example:

```
if(score >= 90)
   cout<<"The grade is A"<<endl;
else if(score >= 80)
      cout<<"The grade is B"<<endl;
else
      cout<<"The grade is C"<<endl;
```

Example 1

- Compute the absolute value of a given number
- AbsoluteValue.cpp

Absolute Value Example

```
int number, temp;
cout << "Please enter an integer: ";
cin >> number;
cout << endl;

temp = number;
if (number < 0)
        temp = -number;

cout << "The absolute value of " << number
         << " is " << temp << endl;
```

Example 2

- Compare two given numbers
- CompareNumbers.cpp

Compare Example

```
int num1, num2, larger;
cout << "Enter any two integers: ";
cin >> num1 >> num2;     cout << endl;

if (num1 > num2) {
        larger = num1;
        cout << "The larger number is " << larger << endl;}
else if (num2 > num1) {
        larger = num2;
        cout << "The larger number is " << larger << endl; }
else
        cout << "Both numbers are equal." << endl;
```

More selection structures

- Conditional operator
- Switch structure

Conditional Operator (?:)

- Conditional operator (?:) takes three arguments (ternary)
- Syntax for using the conditional operator:

  expression1 ? expression2 : expression3

- This evaluates to an expression
- If expression1 is true, the result of the conditional expression is expression2. Otherwise, the result is expression3

Example

- **Example:**

  The statements

```
if(a>=b)   max = a;
else       max = b;
```

  can be expressed, using the conditional operator, as

```
max = (a>=b)? a : b;
```

switch Structure

- Switch structure: alternate to if-else
- Syntax:

```
switch (expression)
    {
     case  value 1: statements1
                      break;
    case value 2: statements2
                     break;
     …
    case value n: statementsn
                     break;
    default : statements
    }
```

- Advice: Use if-else instead of switch

Example

```
switch( grade )  //  grade is a variable of type char
   {
    case 'A':  cout<<"The grade is A";
                  cout <<"!!!";
                  break;
   case 'B':  cout<<"The grade is B";
                  break;
  case 'C': cout<<"The grade is C";
                  break;
  default : cout << "The grade is invalid";
  }
```

Summary

- Control structures alter normal control flow
- Most common control structures are selection and repetition
- Relational operators: ==, <, <=, >, >=, !=
- Logical expressions evaluate to 1 (true) or 0 (false)
- Logical operators: ! (not), && (and), || (or)

Summary

- Two selection structures: one-way selection and two-way selection
- The expression in an if or if...else structure is usually a logical expression
- else is not a standalone statement in C++. Every else has a related if
- A sequence of statements enclosed between braces, { and }, is called a compound statement or block of statements
- More selection structures: conditional operator, switch

Repetition

Why Is Repetition Needed?

- Repetition allows you to efficiently use variables
- Can input, add, and average multiple numbers using a limited number of variables
- For example, to add five numbers:
  - Declare a variable for each number, input the numbers and add the variables together
  - Create a loop that reads a number into a variable and adds it to a variable that contains the sum of the numbers

The while Loop

- The general form of the while statement is:

```
while(expression)
        statement
```

- while is a reserved word
- Statement can be simple or compound
- Expression acts as a decision maker and is a logical expression
- Statement is called the body of the loop
- The parentheses are part of the syntax

The while Loop (continued)

- Expression provides an entry condition
- Statement executes if the expression initially evaluates to true
- Loop condition is then reevaluated
- Statement executes until the expression is no longer true

The while Loop (continued)

- Infinite loop: continues to execute endlessly
- Can be avoided by including statements in the loop body that assure exit condition will eventually be false

Example

- Print the nonnegative multiple of 5 up to 20
- PrintSomeNumbers.cpp

Print multiple of 5 up to 20 example

```
int counter;    //loop control variable
counter = 0;      // initialize counter

while(counter <= 20)
 {
    cout <<counter<<" ";
    counter = counter +5;
 }

cout << endl;
```

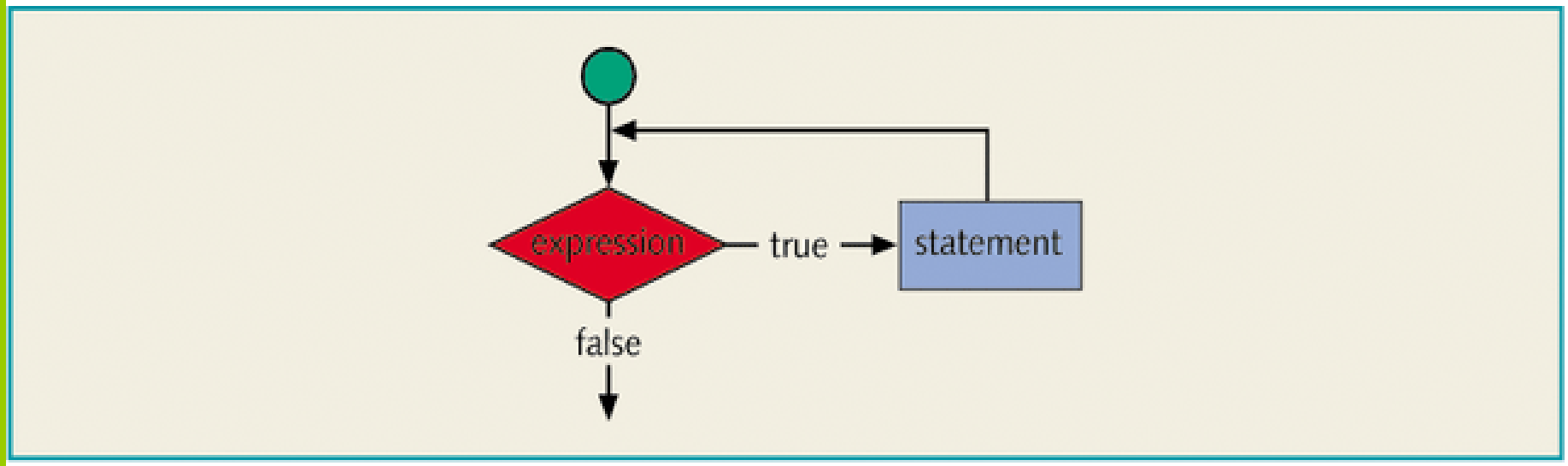

Figure 5-1 `while` loop

Counter-Controlled while Loops

- If you know exactly how many pieces of data need to be read, the while loop becomes a counter-controlled loop
- The syntax is:

```
counter = 0;
while(counter < N)
{
        .
        counter++;
        .
}
```

Example

- Compute the sum and average of a list of integers.

  Ask the user first to enter the number of integers in the list. Then the user enters the integers in the list one by one.

- CountControl.cpp

Compute the sum and average example

```
int limit;        //variable to store the number of items  in the list
int number;                    //variable to store the number
int sum;              //variable to store the sum
int counter;          //loop control variable

cout << "Enter the number of integers in the list: ";
cin >> limit;

sum = 0;   counter = 0;
cout<<"Enter the numbers:"<<endl;
while (counter < limit) {
     cin >> number;
     sum = sum + number;
     counter++;
}
```

Compute the sum and average example (continued)

```
cout << "The sum of the " << limit
        << " numbers = " << sum << endl;

if (counter != 0)
     cout << "The average = "
             << static_cast<double>(sum) / counter << endl;
else
    cout << "No input." << endl;
```

- static_cast<double>(sum): converts sum from int to double
- We can use also sum+0.0 instead of static_cast<double>(sum)

Sentinel-Controlled while Loops

- Don’t know how many entries to be read
- Know that last entry is a special value, called sentinel
- Sentinel variable is tested in the condition and loop ends when sentinel is encountered
- The syntax is:

```
cin>>variable;
while(variable != sentinel)
{
        .
        cin>> variable;
        .
}
```

Example

- Sentinel version of the previous example: compute the sum and the average of a list of number
- SentinelControl.cpp

Sum and the average: sentinel controlled example

```
int number;    //variable to store the number
int sum = 0;   //variable to store the sum
int count = 0;              //variable to store the total numbers read

cout << " Enter numbers ending with " << SENTINEL << endl;

cin >> number;
while (number != SENTINEL) {
        sum = sum + number;
        count++;
        cin >> number;
 }
// rest as  before
```

The for-loop

- Typically used as an alternative of counter-controlled while-loop
- The general form of the for statement is:

```
for(initial statement; loop condition; update statement)
        statement
```

- The initial statement, loop condition, and update statement are called for loop control statements

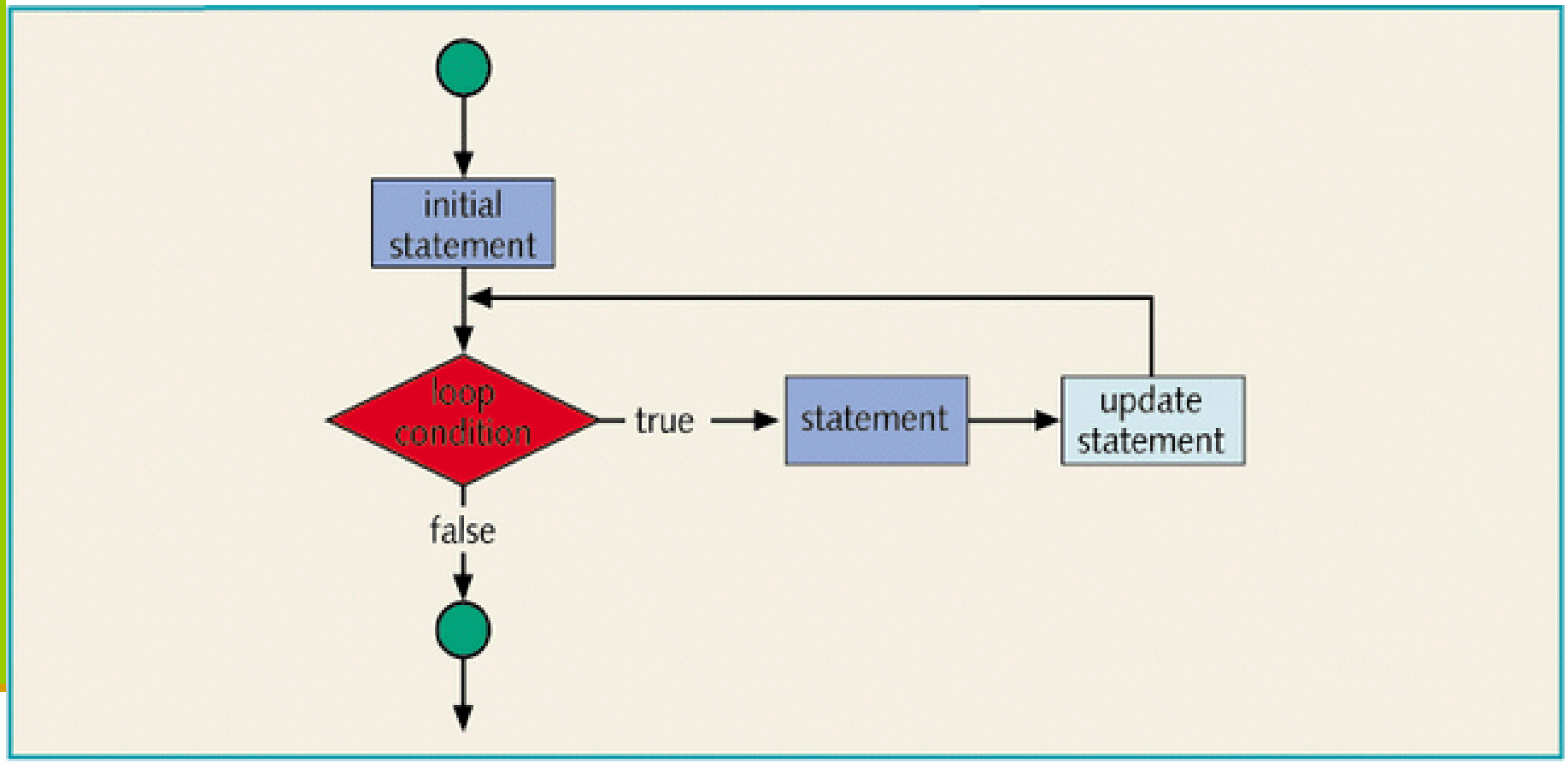

Figure 5-2 `for` loop

The for Loop (continued)

- The for loop executes as follows:
  - initial statement executes
  - loop condition is evaluated
    - If loop condition evaluates to true
      - Execute for loop statement
      - Execute update statement
      - Repeat previous step until the loop condition evaluates to false
- initial statement initializes a variable

The for Loop (continued)

- Use the initial statement to initialize your control variable, as it is first to be executed and is executed only once
- If the loop condition is initially false, the loop body does not execute
- Use the update statement to change the value of the loop control variable which eventually sets the value of the loop condition to false on termination
- The for loop executes indefinitely if the loop condition is always true

Example

- Determine the sum of the first n positive numbers
- SumNNumbers.cpp

Sum of the first n positive numbers: for loop example

```
int counter;                //loop control variable
int sum;                    //variable to store the sum of numbers
int N;     //variable to store the number of  first positive integers to be added

cout << "Enter the number of positive  integers to be added: ";
cin >> N;

sum = 0;
cout << endl;

for (counter = 1; counter <= N; counter++)
      sum = sum + counter;

 cout << "The sum of the first " << N << " positive integers is "
       << sum << endl;
```

Nested Control Structures

- Suppose we want to create the following pattern

```
*
**
***
****
*****
```

- In the first line, we want to print one star, in the second line two stars and so on

Nested Control Structures (continued)

- Since five lines are to be printed, we start with the following for statement

```
for(i = 1; i <= 5 ; i++)
```

- The value of i in the first iteration is 1, in the second iteration it is 2, and so on
- Can use the value of i as limit condition in another for loop nested within this loop to control the number of starts in a line

Nested Control Structures (continued)

- The syntax is:

```
for(i = 1; i <= 5 ; i++)
{
        for(j = 1; j <= i; j++)
                cout<<"*";
        cout<<endl;
}
```

Nested Control Structures (continued)

- What pattern does the code produce if we replace the first for statement with the following?

```
for (i = 5; i >= 1; i--)
```

- That is,

```
for (i = 5; i >= 1; i--)
{
        for(j = 1; j <= i; j++)
                cout<<"*";
        cout<<endl;
}
```

Nested Control Structures (continued)

- Answer:

```
*****
****
***
**
*
```

Using Boolean Variables in Loops: Testing Primality Example

- An integer if n >1 is prime if it has no (positive) divisors other than 1 and n itself
- Given integer n, check if n is prime

```
if (n<=1)  cout<<“not prime”;
else {
      bool isPrime  = true;
      … look for evidence that n is not prime: a divisor of n
      … if divisor found, set isPrime to false
     if(isPrime==true)   // or equivalently: if(isPrime)
            cout<<“prime”;
     else   cout<<“not prime”;
}
```

Using Boolean Variables in Loops: Testing Primality Example (Continued)

```
if (n<=1)  cout<<“not prime”;
else {
      bool isPrime  = true;
      int d = 2;
      while(d<= n-1) {
         if( n%d == 0)      isPrime =false;
         d++;
       }
      if(isPrime) cout<<“prime”; else cout<<“not prime”;
}
```

Using Boolean Variables in Loops: Testing Primality Example (Continued)

Faster Test: stop looking for divisors when you find one

```
if (n<=1)  cout<<“not prime”;
else {
      bool isPrime  = true;
      int d = 2;
      while(d<= n-1 && isPrime == ture) {
       // or equivalently:  while(d<=n-1&& isPrime)
          if( n%d == 0)    isPrime =false;
          d++;
       }
      if(isPrime) cout<<“prime”; else cout<<“not prime”;
}
```

Break & Continue Statements

- break and continue alter the flow of control
- When the break statement executes in a repetition structure, it forces control to exit the structure
- The break statement can be used in while and for loops

Break & Continue Statements (continued)

- The break statement is used for two purposes:
  1. To exit early from a loop
  2. To skip the remainder of the switch structure
- After the break statement executes, the program continues with the first statement after the structure
- The use of a break statement in a loop can eliminate the use of certain (flag) variables

Break & Continue Statements (continued)

- continue is used in while and for structures
- When executed in a loop
  - It skips remaining statements and proceeds with the next iteration of the loop

Break & Continue Statements (continued)

- In a while structure
  - Expression (loop-continue test) is evaluated immediately after the continue statement
- In a for structure, the update statement is executed after the continue statement
  - Then the loop condition executes

Examples (Continued)

- Primality test speedup using the break statement (form Programming Assignment 2):

```
// to check if integer n is prime

bool isPrime = true;
for(int i=2;i*i<=n; i++)
      if( n%i == 0)
      {
            isPrime =false;
            break;
      }

if (n==1) isPrime = false;    // 1 is not prime by convention
if(isPrime) cout <<n<<“  is Prime.”;
else cout <<n<<“  is not Prime.”;
```

Summary

- We studied two repetition structures: while, for, … more later on
- While and for are reserved word
- while: expression is the decision maker, and the statement is the body of the loop
- In a counter-controlled while loop,
  - Initialize counter before loop
  - Body must contain a statement that changes the value of the counter variable

Summary

- A sentinel-controlled while loop uses a sentinel to control the while loop
- for loop: simplifies the writing of a count-controlled while loop
- Nested control structures
- Break and continue statements

Plan

- To study more interesting examples, need to store and manipulate a list of data
- Need Arrays
- Plan:
  - An introduction to arrays
  - Control structures with arrays