

# CSCI 152

## Programming Fundamentals II



Summer 2008

---

Control Structures  
Selection & Looping controls

June 5, 2008

# Objectives

## (Selection Control Structures Ch #4)

---

In this lecture/chapter you should review and:

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

# 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

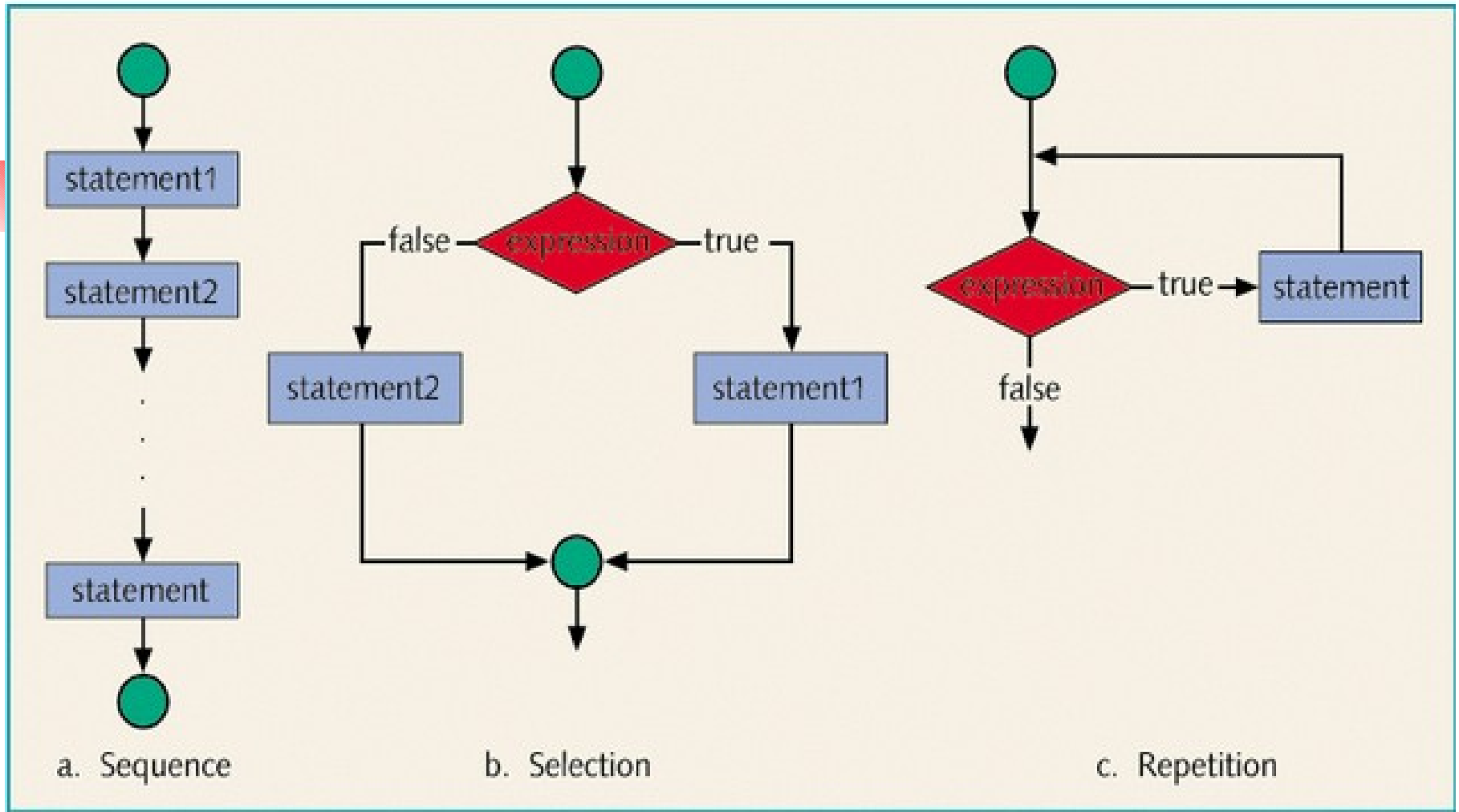


Figure 4-1 Flow of execution

# 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

# One-Way (if) Selection



---

- The syntax of one-way selection is:

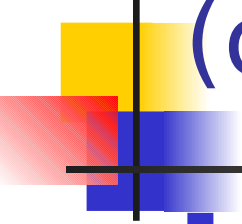
if(expression)

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 usually a logical expression
- The statement is any C++ statement

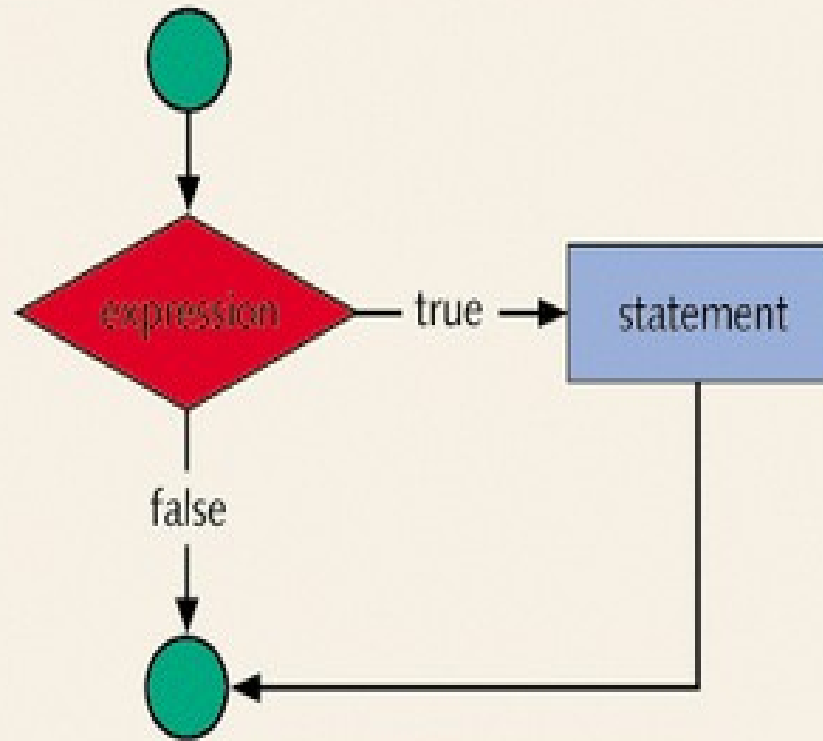


Figure 4-2 One-way selection

# 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

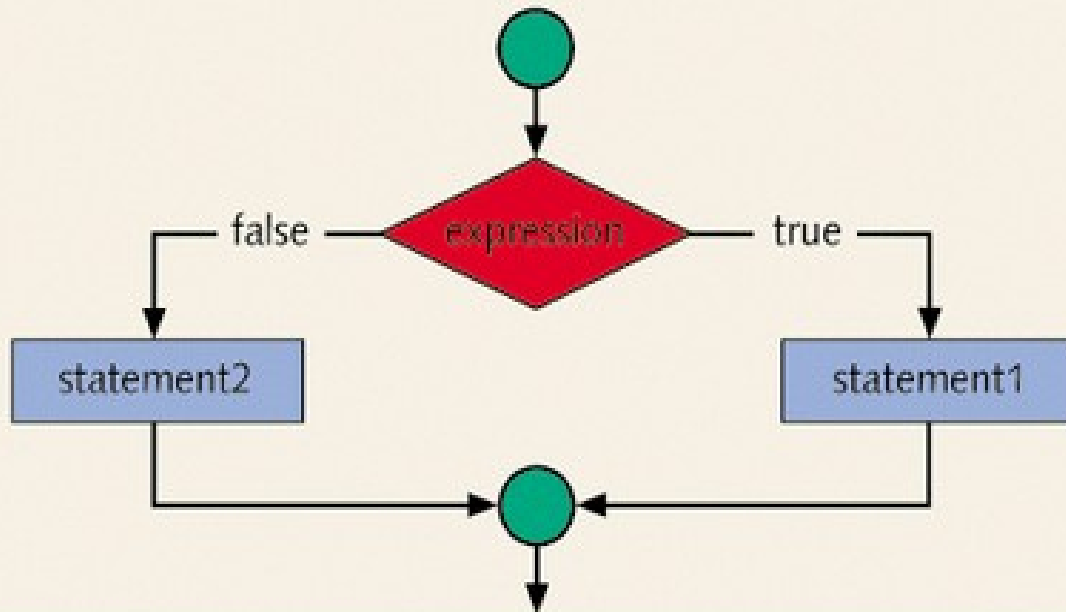


Figure 4-3 Two-way selection

# Compound (Block of) Statement



---

- Compound statement (block of statements):

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

- A compound statement is a single statement

# Compound Statement

## Example

---

```
if(age > 18)
{
    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 F"<<endl;
```



# switch Structures

---

- Switch structure: alternate to if-else
- Switch expression is evaluated first
- Value of the expression determines which corresponding action is taken
- Expression is sometimes called the selector

# switch Structures (continued)



---

- Expression value can be only integral
- Its value determines which statement is selected for execution
- A particular case value should appear only once

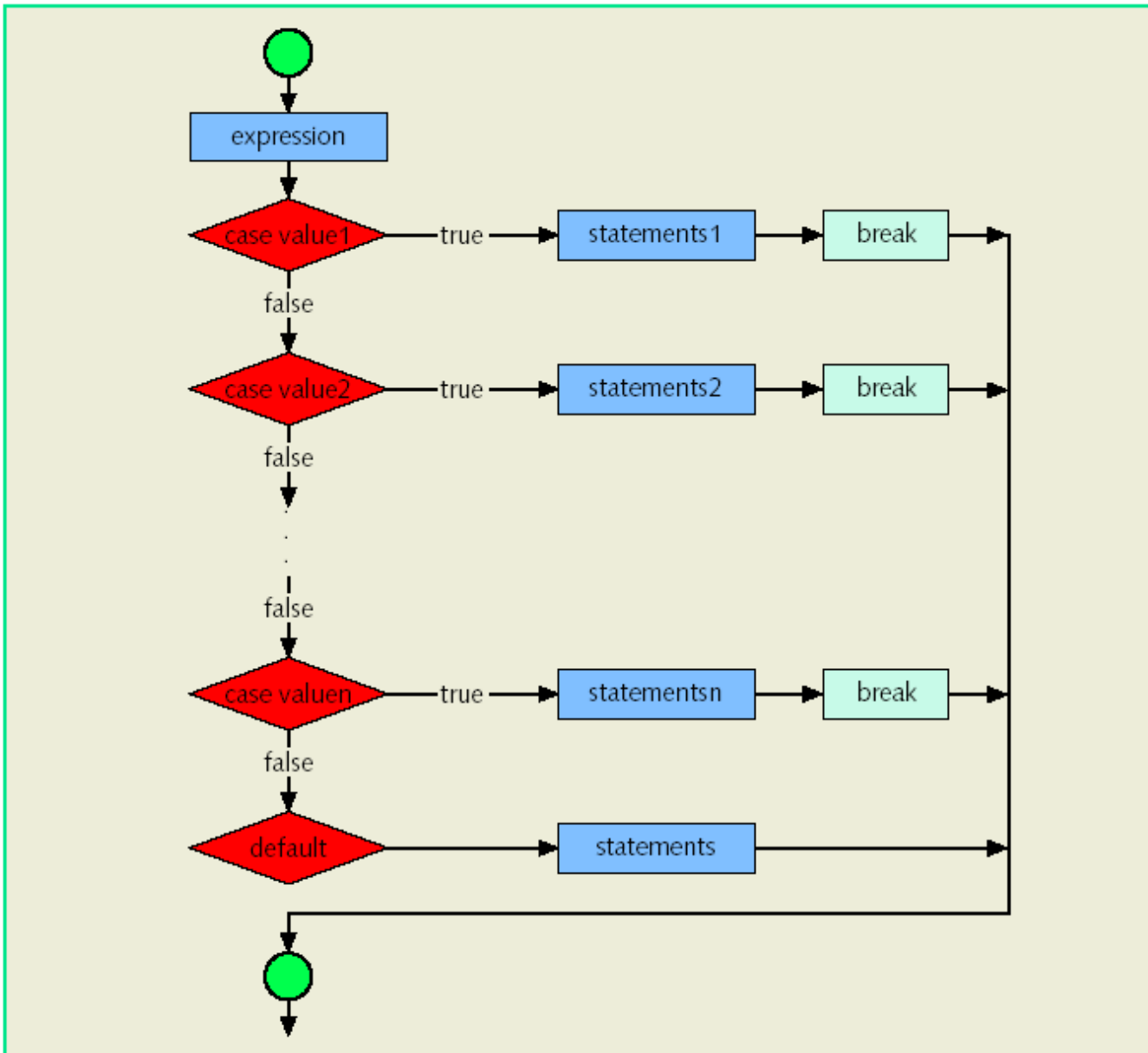


Figure 4-4 `switch` statement

# switch Structures (continued)



---

- One or more statements may follow a case label
- Braces are not needed to turn multiple statements into a single compound statement
- The break statement may or may not appear after each statement
- switch, case, break, and default are reserved words

# switch Statement Rules



---

- When value of the expression is matched against a case value,
  - Statements execute until break statement is found or the end of switch structure is reached
- If value of the expression does not match any of the case values
  - Statements following the default label execute If no default label and no match the entire switch statement is skipped
- A break statement causes an immediate exit from the switch structure



# 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
- A sequence of statements enclosed between braces, { and }, is called a compound statement or block of statements
- Switch structure handles multiway selection

# Objectives

## (Looping Control Structures Ch #5)

---

In this lecture/chapter you should review and:

- Learn about repetition (looping) control structures
- Explore how to construct and use count-controlled, sentinel-controlled, flag-controlled, and EOF-controlled repetition structures
- Examine break and continue statements

# 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 usually a logical expression

# The while Loop (continued)



---

- Expression provides an entry condition
- Statement executes if the expression initially evaluates to true
- Loop condition is then reevaluated
- Statement continues to execute 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

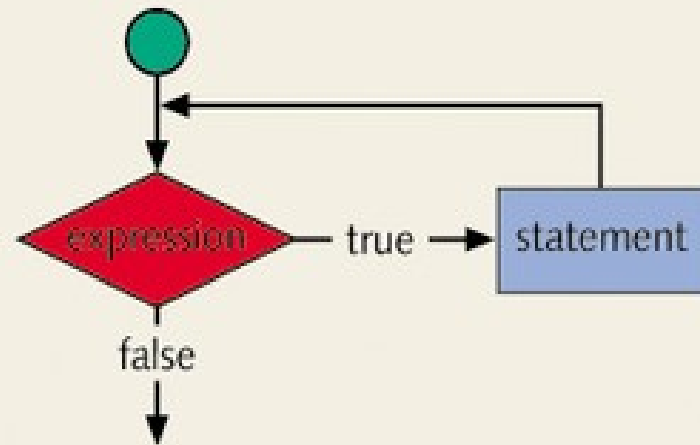


Figure 5-1 while loop

# 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++;
    .
}
```

# Sentinel-Controlled while Loops



---

- Sentinel variable is tested in the condition and loop ends when sentinel is encountered

- The syntax is:

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

# Flag-Controlled while Loops

- A flag-controlled while loop uses a Boolean variable to control the loop
- The flag-controlled while loop takes the form:

```
found = false;
while(!found)
{
.
if(expression)
    found = true;
.
}
```

# EOF-Controlled while

## Loops

- Use an EOF (End Of File)-controlled while loop
- The logical value returned by cin can determine if the program has ended input

- The syntax is:

```
cin >> variable;
while (cin)
{
    .
    cin >> variable;
    .
}
```

# The eof Function



---

- The function eof can determine the end of file status
- Like other I/O functions (get, ignore, peek), eof is a member of data type istream
- The syntax for the function eof is:

`istreamVar.eof()`

where `istreamVar` is an input



# The for 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

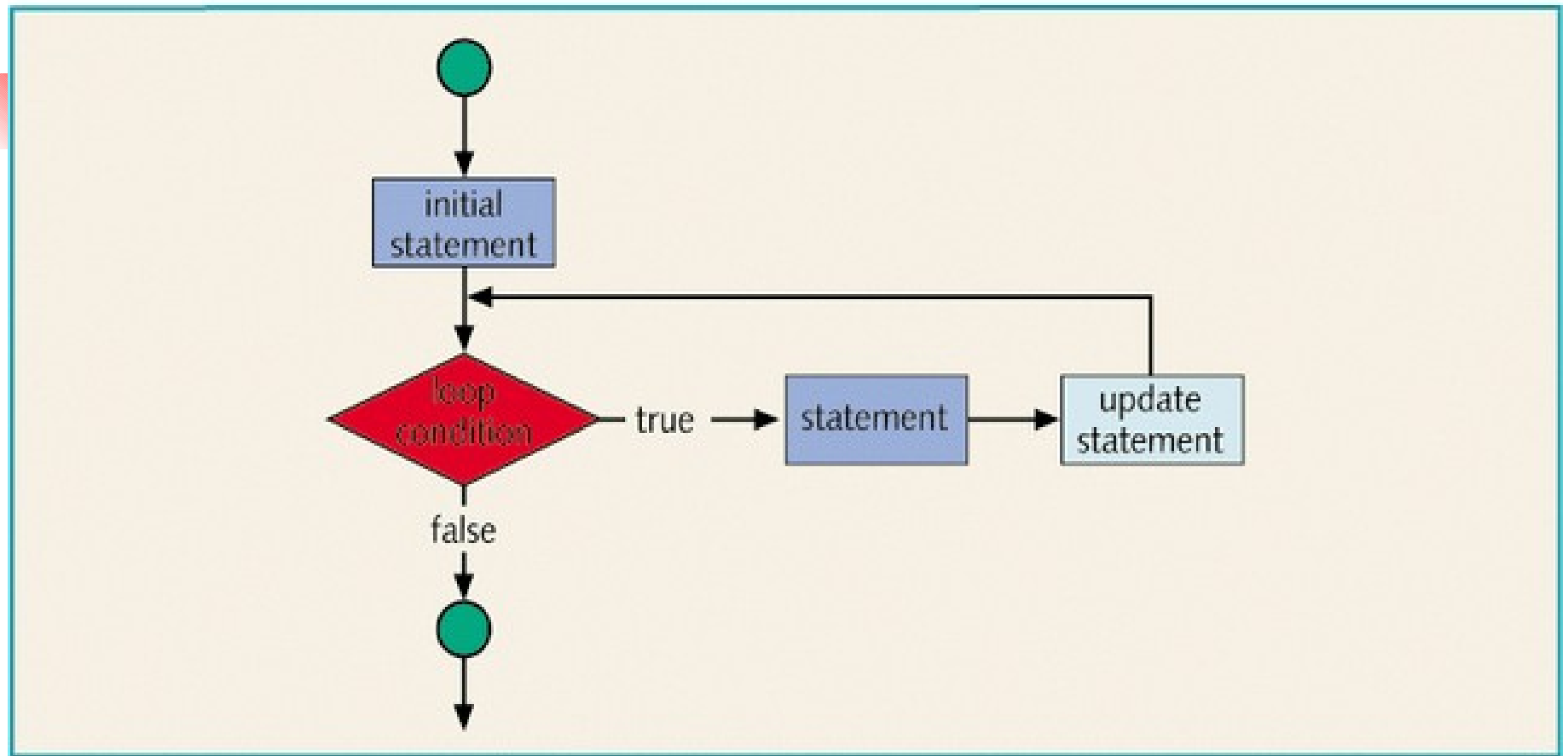


Figure 5-2 for loop

# 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)



---

- initial statement in the for loop is the first to be executed and is executed only once
- If the loop condition is initially false, the loop body does not execute
- The update expression changes the value of the loop control variable which eventually sets the value of the loop condition to false
- The for loop executes indefinitely if the loop condition is always true

# The for Loop (continued)



---

- Fractional values can be used for loop control variables
- A semicolon at the end of the for statement is a semantic error
  - In this case, the action of the for loop is empty
- If the loop condition is omitted
  - It is assumed to be true

# The for Loop (continued)



---

- In a for statement, all three statements (initial statement, loop condition, and update statement) can be omitted
- The following is a legal for loop:

```
for(;;)
```

```
    cout<<"Hello"<<endl;
```

# Summary



---

- C++ has three looping (repetition) structures: while, for, and do...while
- while, for, and do are reserved words
- while and for loops are called pre-test loops