Chapter 4, 5 – Control Structures

Control Structures

- ## Flow carts
  - Illustrates the order events will go in
  - Eq
    - sequential execution:

```
add grade to total        total = total + grade;


add 1 to counter          counter = counter + 1;
```
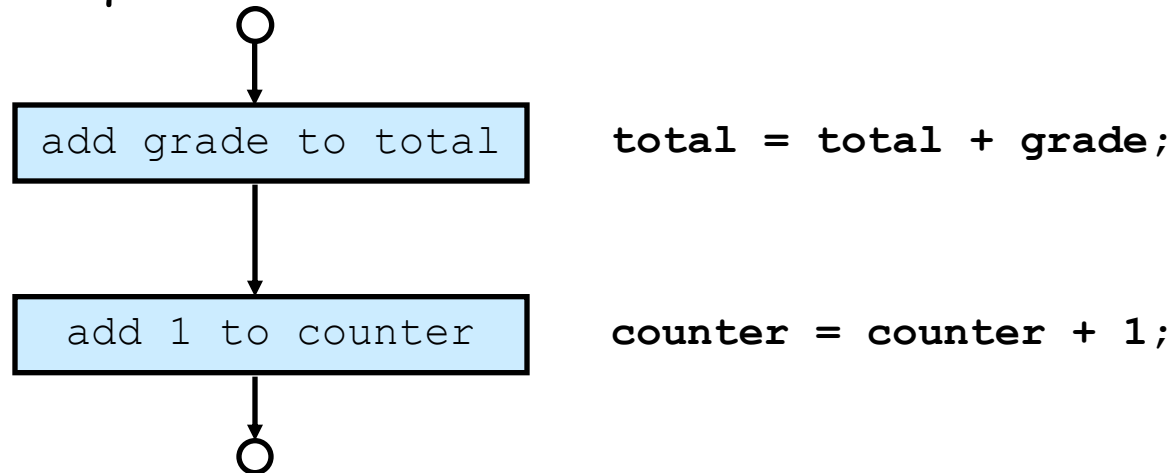
Control Structures

- Program of control
  - Program performs one statement then goes to next line
    - i.e., Sequential execution
  - Different statement other than the next one executes
    - Selection structure
      - The **if** and **if/else** statements
      - The **goto** statement
        - No longer used unless absolutely needed
        - Since it causes many readability problems
    - Repetition structure
      - The **while** and **do/while** loops
      - The **for** and **foreach** loops

If Selection Statement

- Two types of if statement
  - One is without else part
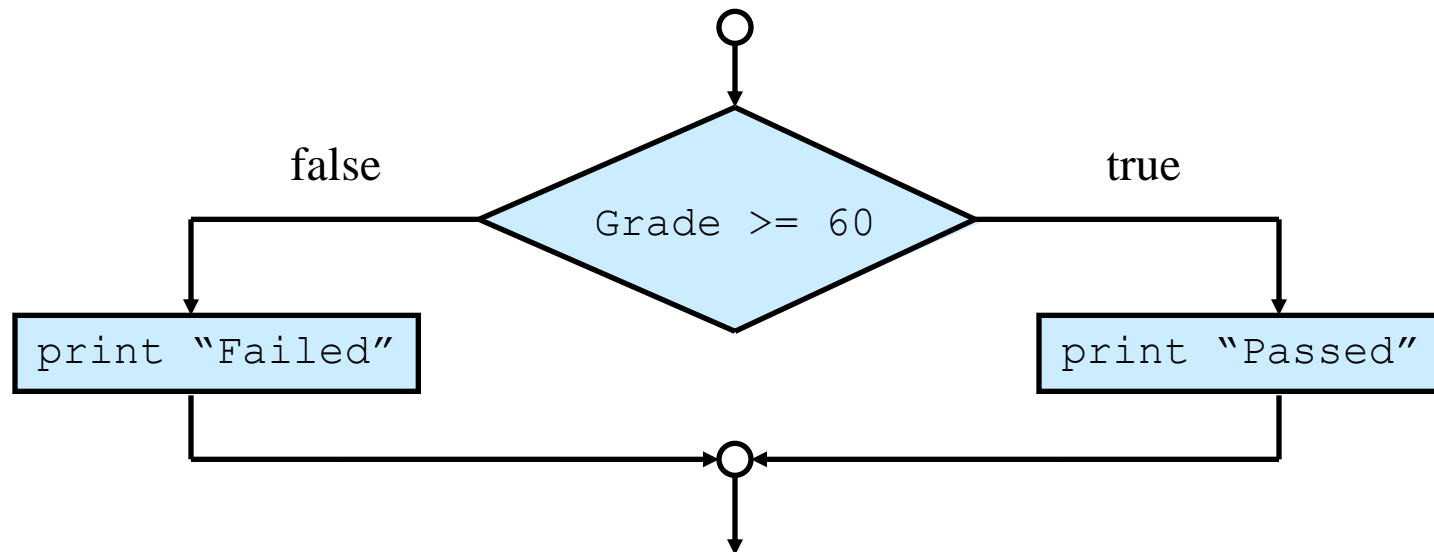 - The other is with else part
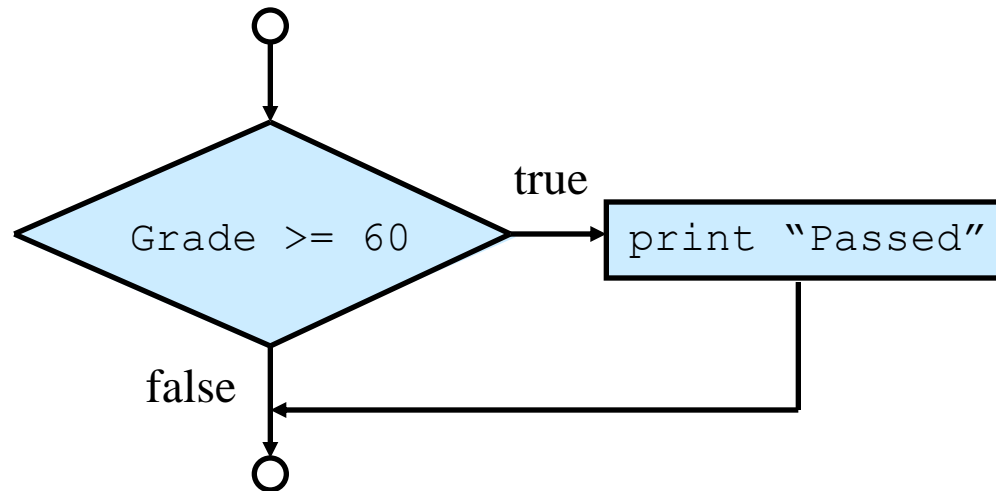

```
if ( studentGrade >= 60 )
    Console.WriteLine( "Passed" );
```

if/else selection statement

- ## The **if/else** structure
  - – Alternate courses can be taken when the statement is false


```
if ( studentGrade >= 60 )
    Console.WriteLine( "Passed" );
else
    Console.WriteLine( "Failed" );
```

Simple/compound statement

- Simple statement do not use brace, "{", "}", such as

        a = a +1;
        if (a >= 1)
            a = a-5 ;

Note:

- In SQL, Pascal, Delphine, ";" is used as a statement separator to separate two statements

  – You can omit the ";" in the last statement in a Pascal program

- In C#, java, ";" is used as a statement terminator to end a statement

  – You cannot omit the " ; " in the last statement in C# or Java program (except the statement in "for" condition test (discussed later))

Simple/compound/Null statement

- In C-like language, "," is used as a statement separator
  - Ex: int i, j, k;
  - for (int i=10, k= 20; i < k; i++)

    ......

    No ";" in the last

- Compound statement
  - containing more than one statement, braced by a pair of parenthesis (no more ";" is needed in the end)
  - A compound statement is considered as a statement, eq:

    if (a >= 1)

    {a = a-5 ; b = b+5; }

    else {a = a+5; b = b-5; }

- Null statement
  - Denoted as ";"
  - Eq: if (a >= 1)
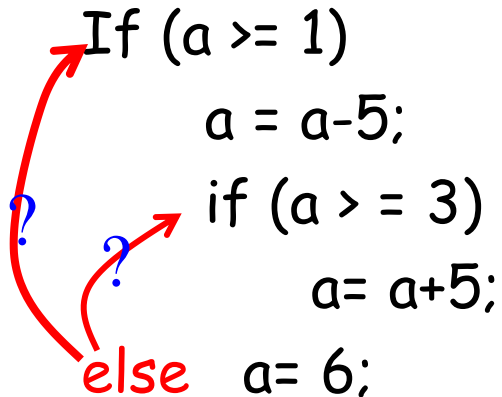
    ;

    else {a = a+5; b = b-5;}

Dangling else

- Dangling else problem (who is the mate of the else)

```
If (a >= 1)
        a = a-5;
        if (a > = 3)
               a= a+5;
     else   a= 6;
```
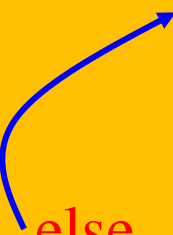
- Which "if" does the "else" hang ?
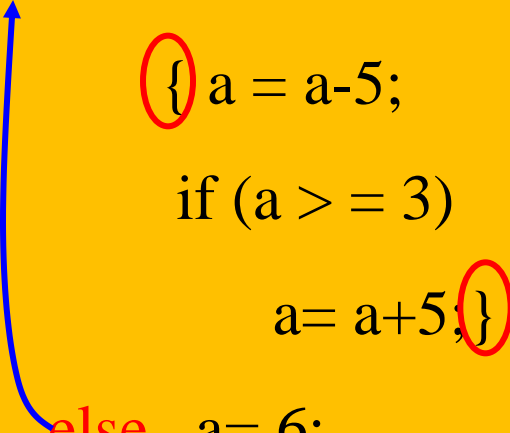
Simple/compound statement

- In C# and Java, the else is default to hang up the nearest " if " , unless you change it

If (a >= 1)

    a = a-5;

    if (a > = 3)

        a= a+5;

else   a= 6;

If (a >= 1)

    { a = a-5;

    if (a > = 3)

        a= a+5;}

else   a= 6;

Change the hang-up rule by adding braces

Orthogonal property in Programming

- A set of properties/statements can be mixed if they adhere their individual definition
  - Then, its effect is still valid, not disturbed by their interaction
- Eq:

```
If (a >= 1)
    { a = a-5;

        if (a > = 3)

            a= a+5; }

else

    { a= 6;

        while (a < 30)

            a= a+10;

    }
```

Simple Conditional statement (?:)

- What is expression?
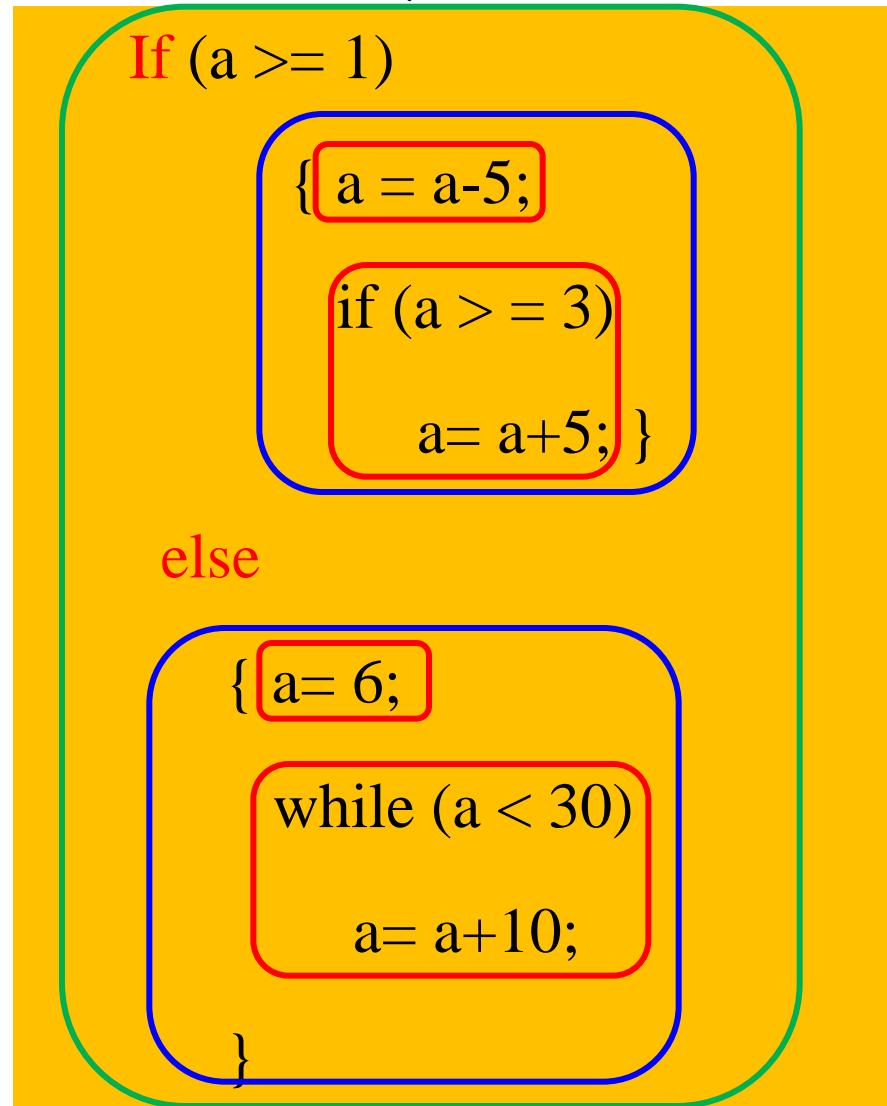 - A value is output, like 4+7, 5/2
- What is conditional express?
  - The output of an expression is true or false only
- What is Simple Conditional expression (?:)
  - An expression with if/else structure
  - The syntax is:

(boolean value) ? Expression1 when true: expression2 when *false*

  - Example:

    b = (a>b)? 10: 8;

```
// if-else construction.
if (input > 0)
    classify = "positive";
else
    classify = "negative";
```
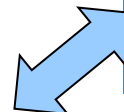
```
classify = (input > 0) ? "positive" : "negative";
```

while Repetition Structure

- While statement
  - The statement in "while" is to be repeated
    - Continues while condition is true
    - Ends when condition is false
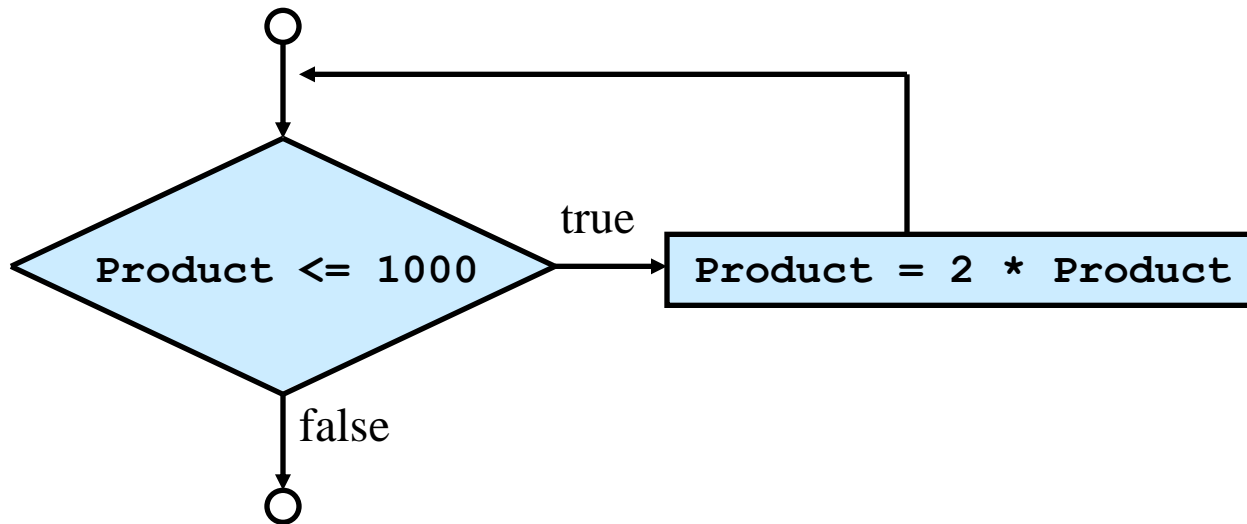
While (Product <= 1000)
    Product = 2 * Product;

```csharp
4   using System;
6   class Average1
7   {
8       static void Main( string[] args )
9       {
10          int total, gradeCounter, gradeValue, average;
16          total = 0;
17          gradeCounter = 1;
20          while ( gradeCounter <= 10 )
21          {
23              Console.Write( "Enter integer grade: " );
26              gradeValue = Int32.Parse( Console.ReadLine() );
29              total = total + gradeValue;
32              gradeCounter = gradeCounter + 1;
33          }
36          average = total / 10;
39          Console.WriteLine( "\nClass average is {0}", average );
41      }
43  }
```

```
Enter integer grade: 100
Enter integer grade: 88
Enter integer grade: 93
Enter integer grade: 55
Enter integer grade: 68
Enter integer grade: 77
Enter integer grade: 83
Enter integer grade: 95
Enter integer grade: 73
Enter integer grade: 62

Class average is 79
```

Legend of the for Repetition Structure

- What is for loop? Why we need it?
  - For loop can directly map to math summation equation

$$\sum_{k=1}^{n} 1 \quad \Longleftrightarrow$$

```
sum = 0;

for (int k = 1; k <= n; k++)

        sum = sum +1;
```

For statement

- For is a counter-controlled repetition, it requires:
  – Define control variables (loop counter)
  – **Initial value** of control variable (executed only once)
  – **Testing condition** (if fail, exit the loop)
  – **Increment/decrement of variables in** each loop

Statement terminator

No terminator (;)

for
keyword

Control variable
name

Final value
of control variable

```
for (    int    counter =    1; counter <=    10; counter++ )
```

Initial value
of control variable

Increment
of control variable

Loop-continuation condition
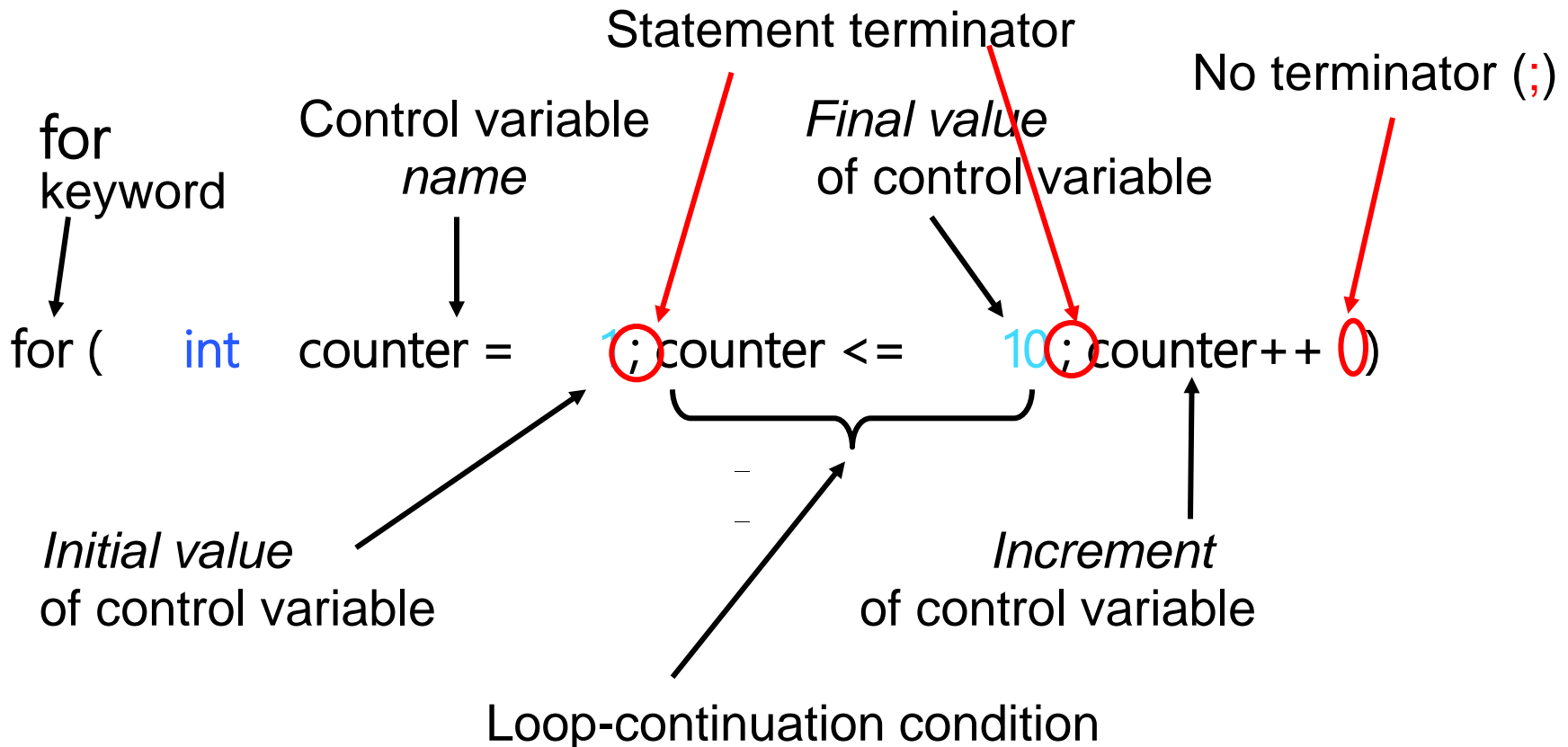
For loop

- **for (**statement*1;* conditional *expression;* statement*3* **)**
  statement2;

```
            ○
            │
            ▼
┌───────────────────────┐
│      Statement 1      │
└───────────────────────┘
            │
            ▼
      ◇ Conditional
        expression ◇ ───── true ──→ ┌──────────────────┐
            │                       │   Statement 2    │
          false                     └──────────────────┘
            │                              │
            ○                              ▼
                              ┌──────────────────┐
                              │   Statement 3    │
                              └──────────────────┘
```
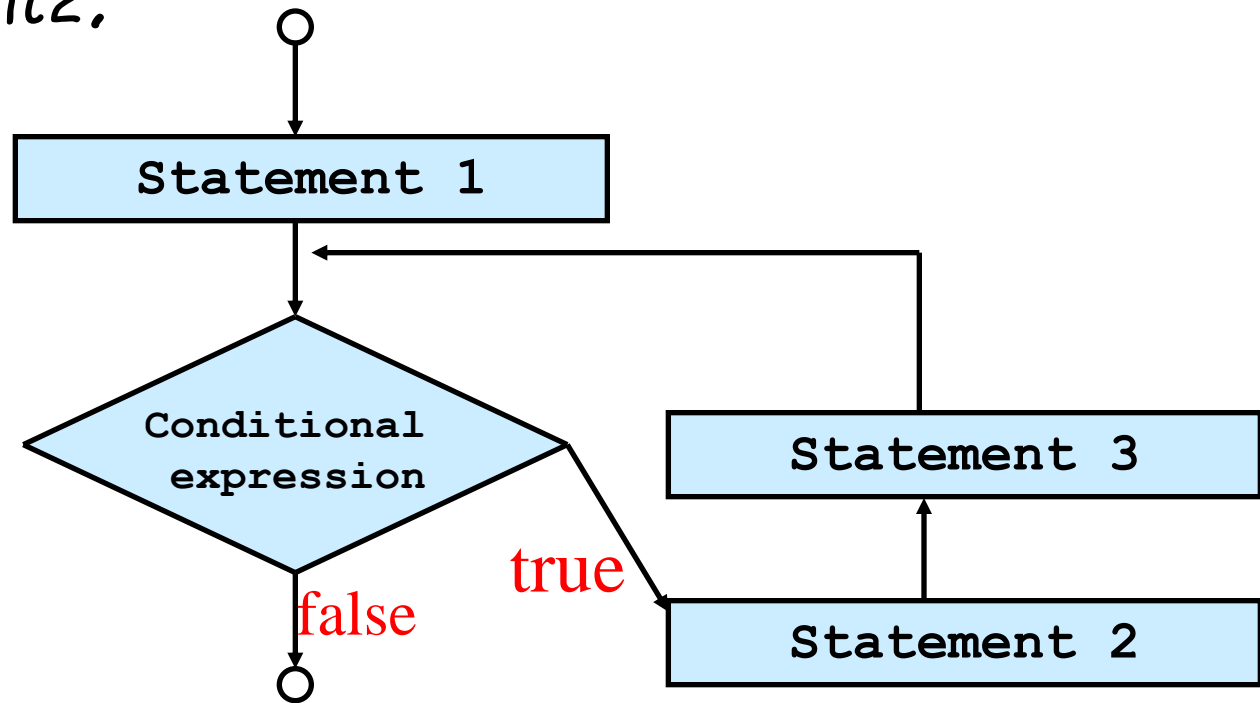
- **Eq:**
  for (int i = 1, sum = 0; i <= 100; i++)
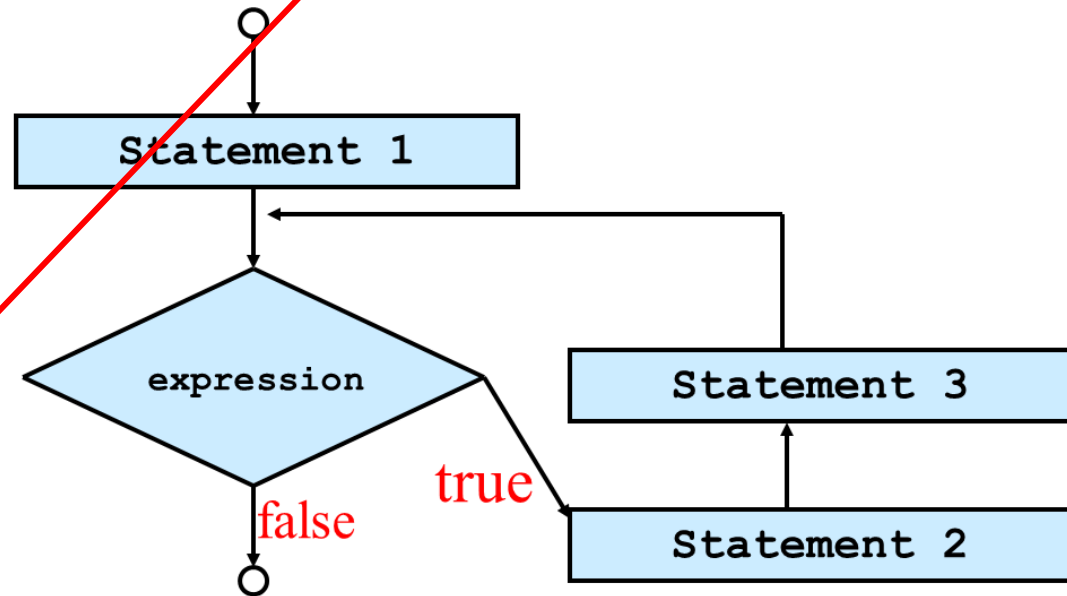   **sum = sum + i;**

The for Repetition Structure (cont.)

for (statement*1;* expression*;* statement*3***)**
   statement*2;*

can be rewritten as:

statement*1;*
while (expression**) {**
   statement*2;*
   statement*3;*
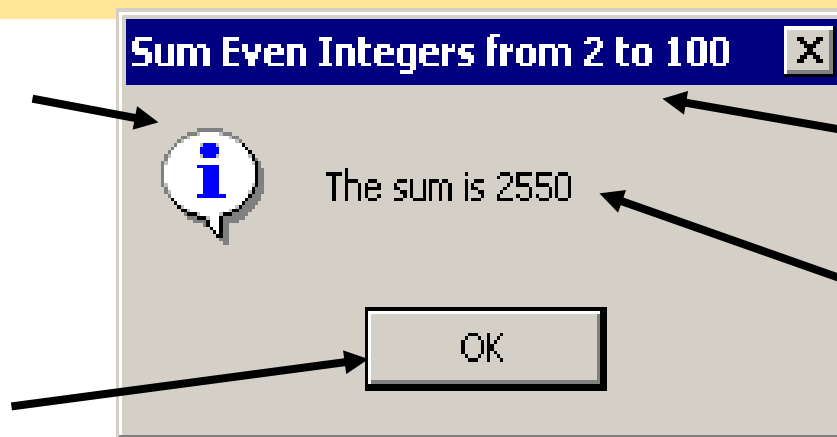**}**

```csharp
4    using System;
5    using System.Windows.Forms;
7    class Sum
8    {
9        static void Main( string[] args )
10       {
11           int sum = 0;
13           for (int number= 2; number <= 100; number += 2 )
14               sum += number;
16           MessageBox.Show( "The sum is " + sum,
17               "Sum Even Integers from 2 to 100",
18               MessageBoxButtons.OK,
19               MessageBoxIcon.Information );
21       }
23   }
```

Argument 4:
MessageBox
Icon (Optional)

Sum Even Integers from 2 to 100   ✕

ⓘ   The sum is 2550

OK

Argument 2: Title bar string (Optional)

Argument 1: Message to display

Argument 3: **OK** dialog button. (Optional)

Assignment Operators

- Assignment expression abbreviations

  c = c + 3;

  can be abbreviated as

  c += 3;

- Statements of the form

  variable = variable operator expression;

  can be rewritten as

  variable operator= expression;

- Examples of other assignment operators include:

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

Message boxes

- Buttons
  - **OK**
  - **OKCancel**
  - **YesNo**
  - **AbortRetryIgnore**
  - **YesNoCancel**
  - **RetryCancel**

- Icons
  - **Exclamation**
  - **Question**
  - **Error**
  - **Information**

Important Message

Dot Net Perls is awesome.

OK

Important Question

Is Dot Net Perls awesome?

Yes | No

Important Query

Is Dot Net Perls awesome?

Yes | No | Cancel

Critical Warning

Dot Net Perls is the best.

OK | Cancel | Help
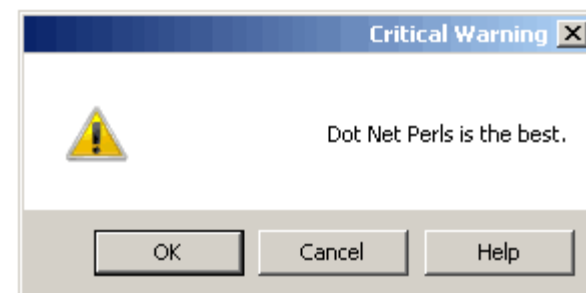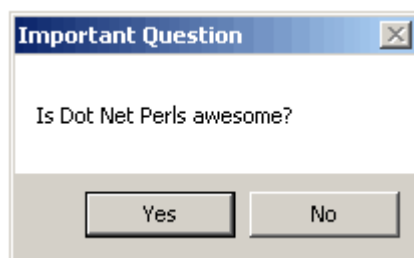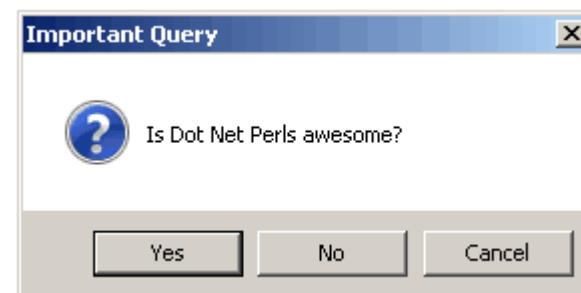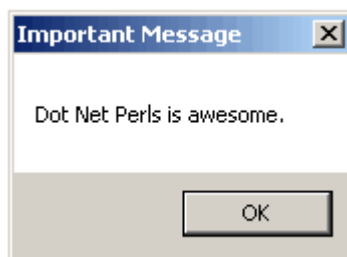
MessageBox Icons	Icon	Description
`MessageBoxIcon.Excl-amation`		Displays a dialog with an exclamation point. Typically used to caution the user against potential problems.
`MessageBoxIcon.Info-rmation`		Displays a dialog with an informational message to the user.
`MessageBoxIcon.Ques-tion`		Displays a dialog with a question mark. Typically used to ask the user a question.
`MessageBoxIcon.Erro-r`		Displays a dialog with an **x** in a red circle. Helps alert user of errors or important messages.
Fig. 5.6 Icons for message dialogs.		

MessageBox Buttons	Description
MessageBoxButton. OK	Specifies that the dialog should include an **OK** button.
MessageBoxButton. OKCancel	Specifies that the dialog should include **OK** and **Cancel** buttons. Warns the user about some condition and allows the user to either continue or cancel an operation.
MessageBoxButton. YesNo	Specifies that the dialog should contain **Yes** and **No** buttons. Used to ask the user a question.
MessageBoxButton. YesNoCancel	Specifies that the dialog should contain **Yes**, **No** and **Cancel** buttons. Typically used to ask the user a question but still allows the user to cancel the operation.
MessageBoxButton. RetryCancel	Specifies that the dialog should contain **Retry** and **Cancel** buttons. Typically used to inform a user about a failed operation and allow the user to retry or cancel the operation.
MessageBoxButton. AbortRetryIgnore	Specifies that the dialog should contain **Abort**, **Retry** and **Ignore** buttons. Typically used to inform the user that one of a series of operations has failed and allow the user to abort the series of operations, retry the failed operation or ignore the failed operation and continue.

Fig. 5.7   Buttons for message dialogs.

break and continue in loop Statements

- **break** (讓我們到此結束吧!)
  - Used to exit the loop it resides (no more execute the loop the break statement resides),
- **continue** (讓我們從新開始,好嗎?)
  1. skip the rest of the statements after the continue statement in the loop the continue statement resides,
  2. then begin that loop from the first statement of that loop

```csharp
4      using System;
5      using System.Windows.Forms;
7      class BreakTest
8      {
9          static void Main( string[] args )
10         {
11             string output = "";
12             int count;
14             for ( count = 1; count <= 10; count++ )
15             {
16                 if ( count == 5 )
17                     break;
20                 output += count + " ";
22             }
24             output += "\nBroke out of loop at count = " +
                           count;
26             MessageBox.Show( output, "Demonstrating the
                   break statement", MessageBoxButtons.OK,
                   MessageBoxIcon.Information );
29         }
31     }
```

BreakTest.cs
Program Output

Demonstrating the break statement

1 2 3 4
Broke out of loop at count = 5

OK
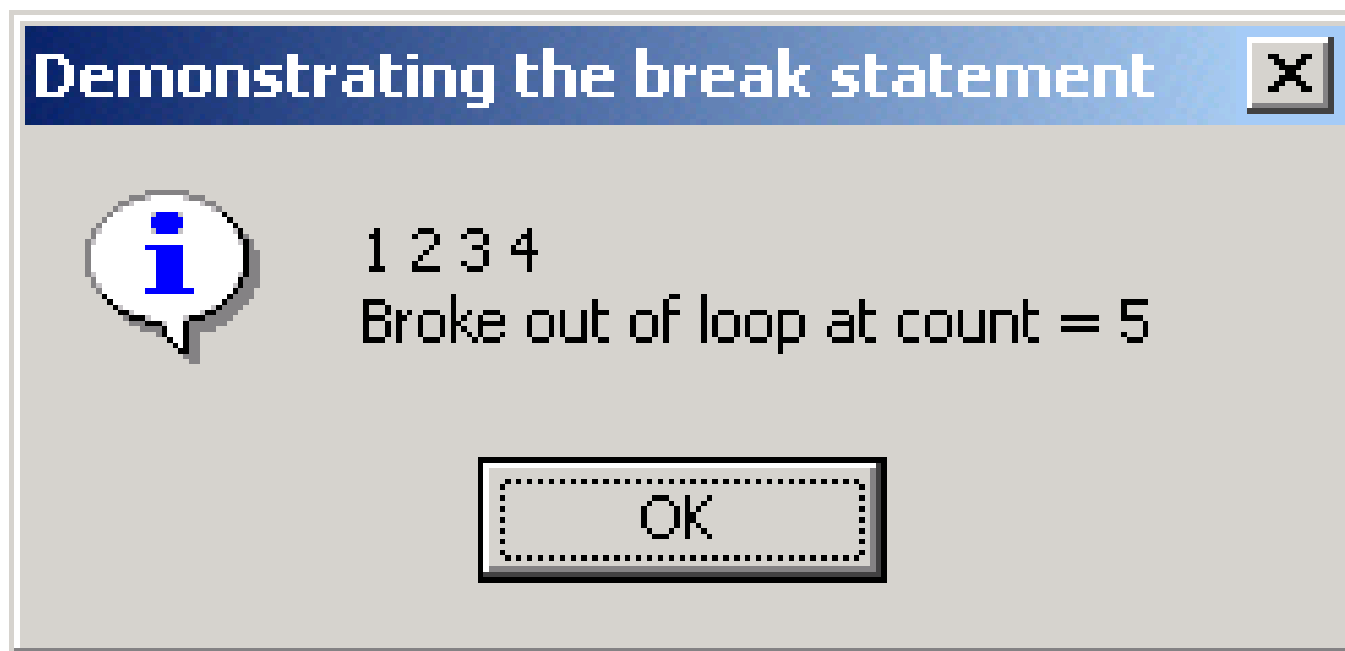
```csharp
 4    using System;
 5    using System.Windows.Forms;
 7    class ContinueTest
 8    {
 9        static void Main( string[] args )
10        {
11            string output = "";
13            for ( int count = 1; count <= 10; count++ )
14            {
15                if ( count == 5 )
16                    continue;
19                output += count + " ";
20            }
22            output += "\nUsed continue to skip printing 5";
24            MessageBox.Show( output, "Using the continue
                    statement", MessageBoxButtons.OK,
                    MessageBoxIcon.Information );
27        } // end method Main
29    } // end class ContinueTest
```

**ContinueTest.cs
Program Output**

Using the continue statement ✕

1 2 3 4 6 7 8 9 10
Used continue to skip printing 5

OK

4.12   Increment and Decrement Operators

- Increment operator
  - Used to add 1 to the variable
  - x++;
  - Same as x = x + 1;
- Decrement operator
  - Used to subtract 1 from the variable
  - y--;
  - Same as y = y-1;
- Pre-increment vs. post-increment
  - x++ or x--
    - Will return the value of x and then add 1 to x (or subtract 1 from x)
  - ++x or --x
    - Will add 1 to x (or subtract 1 from x) and then return the value of x

```csharp
using System;

class Increment
{
    static void Main( string[] args )
    {
        int c;

        c = 5;
        Console.WriteLine( c );      // 5
        Console.WriteLine( c++ );    // 5
        Console.WriteLine( c );      // 6

        Console.WriteLine();

        c = 5;
        Console.WriteLine( c );      // 5
        Console.WriteLine( ++c );    // 6
        Console.WriteLine( c );      // 6

    } // end of method Main

} // end of class Increment
```

Break vs Continue in nested for


```
….
For (….)  {
     …..;
        For  (….)  {
              ……;
              if (….)
                  break;
              ….;
        }
     …….;
}
```

```
….
For (….)  {
     …..;
        For  (….)  {
              ……;
              if (….)
                  continue;
              ….;
        }
     ……;
}
```
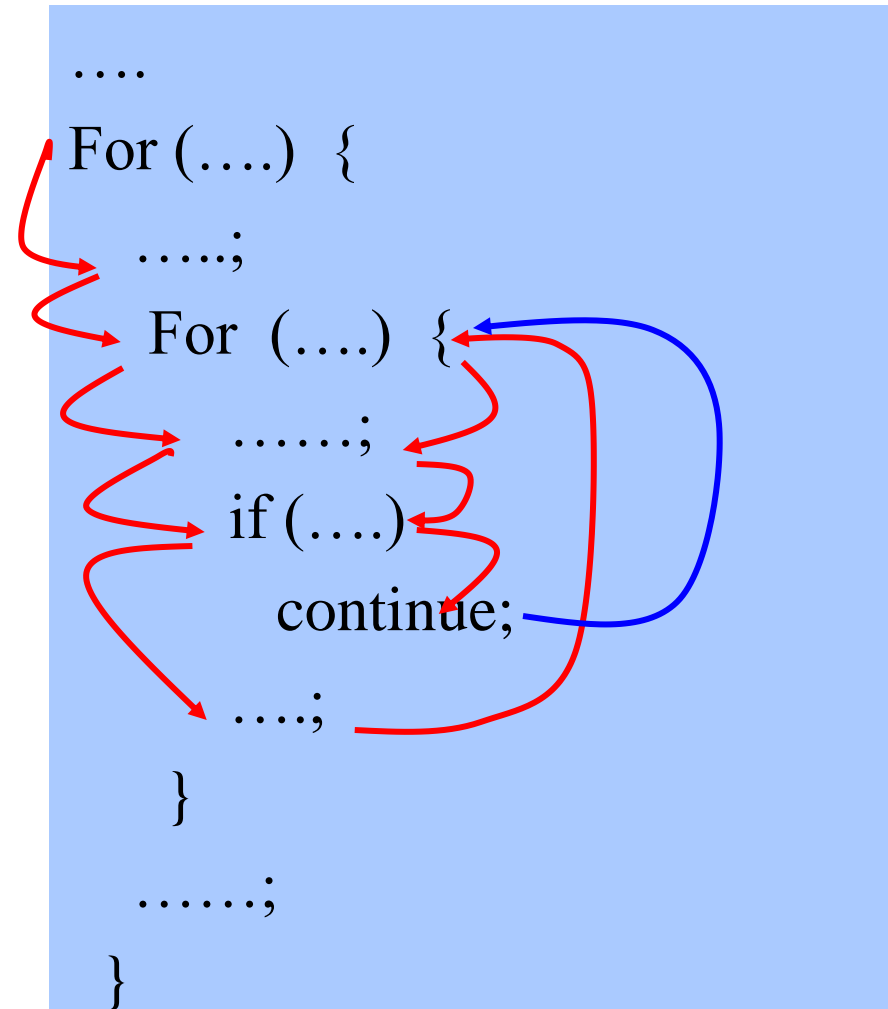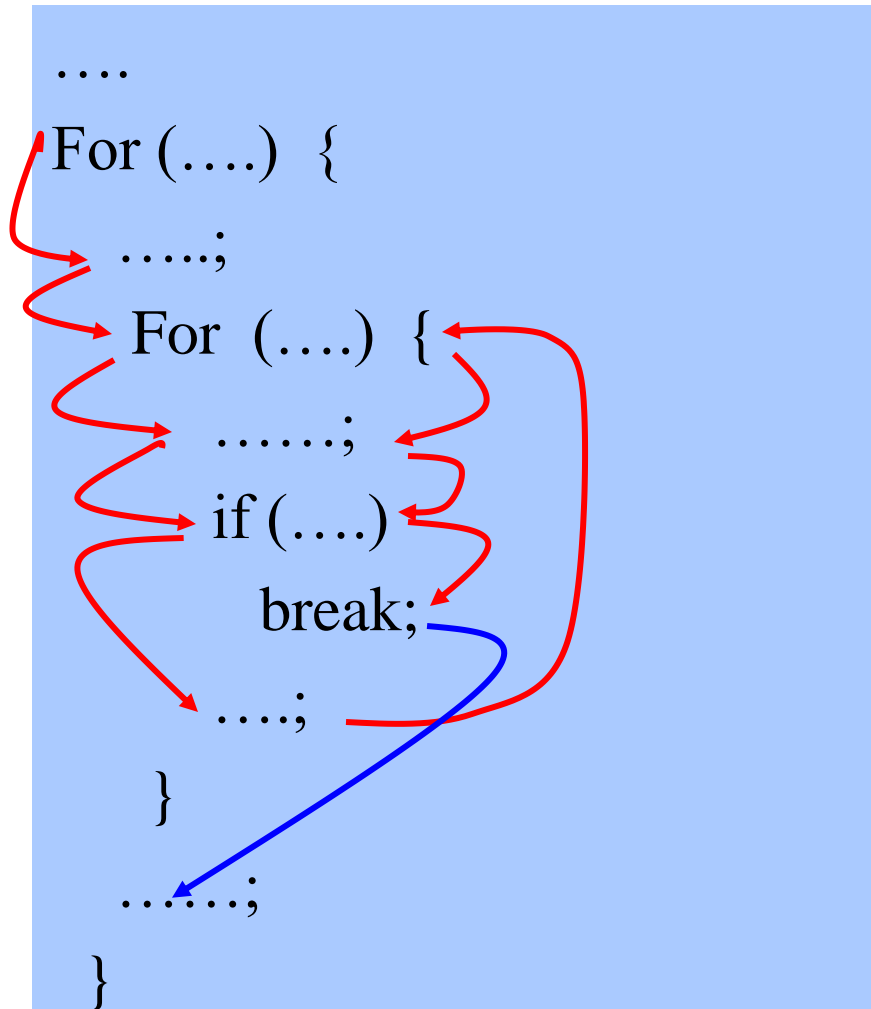
Legend of The for Repetition Structure

- Evolution (in C# is ok; but in Cobol, fortran is not allowed):

Q1: Can the counter be more than one variable?

A1: Yes! Separate by the comma

Q2: Can the incremental expression contain more than one statement?

A2: Yes! Separate by the comma

Q3: Can the main statement contain more than one?

A3: Yes! Use the compound statement

Q4: Can the condition expression be complex?

A4: Yes! By conjunction (&&) or disjointion (||)

Ex. For (i =1, k = 0; i < 10 && k < -4; i++, k--)
        {k+= i; i= k-2;}

5.9   Logical Operators

- Logical operators
  - Allows for forming more complex conditions
  - Combines simple conditions
- C# logical operators
  - **&&**   (logical AND)
  - **||**   (logical OR)
  - **!**   (logical NOT)
  - **&**   (boolean logical AND) (discussed later)
  - **|**   (boolean logical inclusive OR) (discussed later)
  - **^**   (boolean logical exclusive OR) (discussed later)

Legend of The for Repetition Structure

- Q1: Can we change the counter value in the for loop?
  - for (int i = 7; i <= 77; i += 7)

    i= i* k;

  // legal in most lang., but cannot tell how many times the loop runs!

  // In such a case, while statement is used instead (suggestion)

- Q2: Can we use the counter value after the for loop?
  - Some language (like Java) need the counter defined in the for loop,
    - When exited the loop, the counter is undefined
      - It strictly defines the counter being a counter, nothing else.
  - Some language allows the counter can be used after the for loop (I do not suggest)
  - for (i = 7, k = 0; i <= 77; i += 7)

    k ++;

    w= i+10;

Legend of The for Repetition Structure

- we can extend condition test unrelated to counter
  - for (i = 7, k = 0; i <= 77 && w < 44; i += 7)
    k = k + i;

  // legal in most lang., but cannot tell how many times the loop runs!

  // In such a case, while statement is used instead (suggestion)

- Statement in for loop can be degraded to nothing
  - for (i = 7, k = 0; i <= 77; i += 7)
    k ++;

  - Equal to: for (i = 7, k = 0; i <= 77; i += 7, k++) ;
    // using null statement; smart but not legible

```csharp
using System;
using System.Windows.Forms;

class Interest
{
    static void Main( string[] args )
    {
        decimal amount, principal = ( decimal ) 1000.00;
        double rate = .05;
        string output;
        output = "Year\tAmount on deposit\n";
        for ( int year = 1; year <= 10; year++ )
        {
            amount = principal *
                ( decimal ) Math.Pow( 1.0 + rate, year );
            output += year + "\t" +
                String.Format( "{0:C}", amount ) + "\n";
        }
        MessageBox.Show( output, "Compound Interest",
                MessageBoxButtons.OK,
                MessageBoxIcon.Information );
    }
}
```

Interest.cs
Program Output

Compound Interest ✕

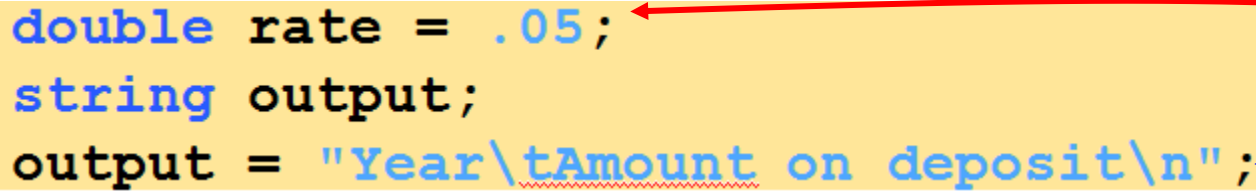| Year	Amount on deposit
1	$1,050.00
2	$1,102.50
3	$1,157.63
4	$1,215.51
5	$1,276.28
6	$1,340.10
7	$1,407.10
8	$1,477.46
9	$1,551.33
10	$1,628.89

OK
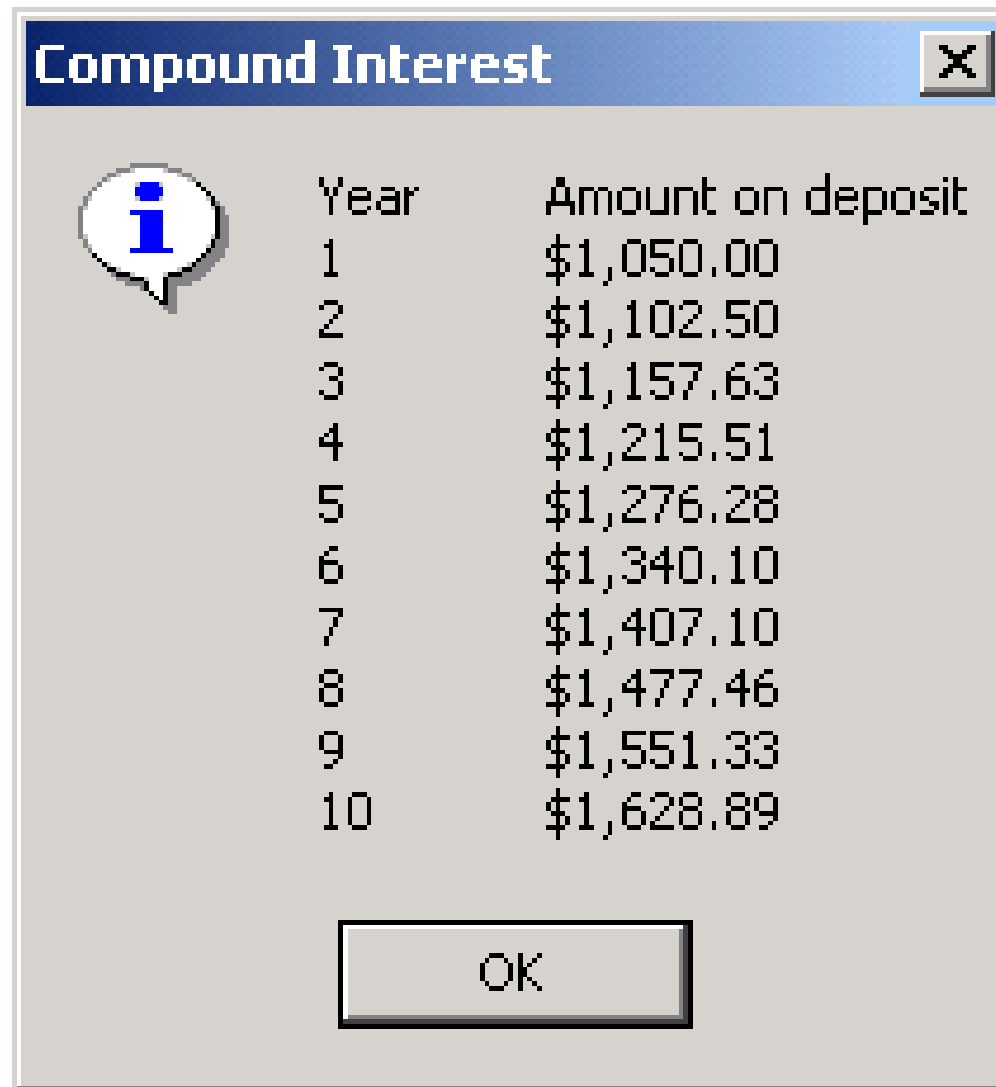
Initialization vs. Assignment

```
double rate = .05;
string output;
output = "Year\tAmount on deposit\n";
```

- What is the difference of initialization and assignment?
  - Initialization of variables is done in compiler time
    - Setting of constant variable is only allowed in initialization, no allowed through assignment (or only once by assignment)
  - Assignment of variables is done in execution time
- The default value of variable if without initialization
  - For integer, floating, its default value is 0
  - For char, it is ' '
  - For string, it is ""

Primitive Data Types (primitive variable)

```
decimal amount, principal = ( decimal ) 1000.00;
double rate = .05;
string output;
output = "Year\tAmount on deposit\n";
```

- Primitive data types
  - "building blocks" for more complicated types
    - Ex. int, float, char, byte, ….
  - No need to "new" the memory for it (discussed later)
  - three boxes for the variable
    - The memory (i.e., third box) to store the variable has existed when the variable is defined

name   attribute   Data memory
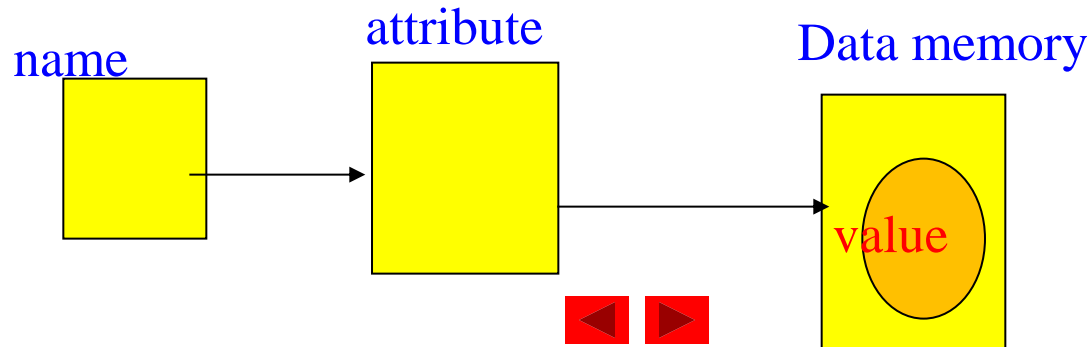
value

Primitive Data Types

- Ex: int k;

  k = 8;

Memory after declaring primitive variable k:

name | attribute | memory
k → integer → 0

Memory after executing the assignment of k:

name | attribute | memory
k → integer → 8
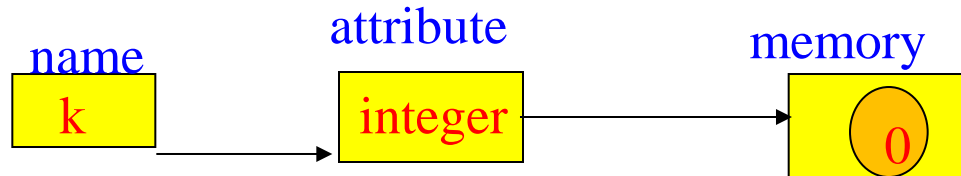
Type	Size in bits	Values	Standard
`boolean`		`true` or `false`	
		[*Note:* The representation of a boolean is specific to the Java Virtual Machine on each computer platform.]	
`char`	16	`'\u0000'` to `'\uFFFF'` (0 to 65535)	(ISO Unicode character set)
`byte`	8	−128 to +127 (-2^7 to $2^7 - 1$)	
`short`	16	−32,768 to +32,767 (-2^{15} to $2^{15} - 1$)	
`int`	32	−2,147,483,648 to +2,147,483,647 (-2^{31} to $2^{31} - 1$)	
`long`	64	−9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (-2^{63} to $2^{63} - 1$)	
`float`	32	*Negative range:* −3.4028234663852886E+38 to−1.40129846432481707e−45 *Positive range:* 1.40129846432481707e−45 to 3.4028234663852886E+38	(IEEE 754 floating point)
`double`	64	*Negative range:* −1.7976931348623157E+308 to −4.94065645841246544e−324 *Positive range:* 4.94065645841246544e−324 to 1.7976931348623157E+308	(IEEE 754 floating point)

primitive types.

```
11        decimal amount, principal = ( decimal ) 1000.00;
12        double rate = .05;
```

```
19            amount = principal *
20                ( decimal ) Math.Pow( 1.0 + rate, year );
```
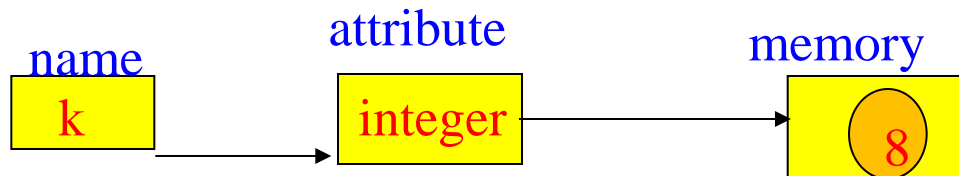
- decimal keyword
  - a 128-bit data type
  - has more precision and a smaller range for financial and monetary calculations than floating point
  - Syntax:
    - decimal myMoney = 99.9m;
    - double x = (double) myMoney;
    - myMoney = (decimal) 30;

attribute

Data memory(16 bytes)

name

value
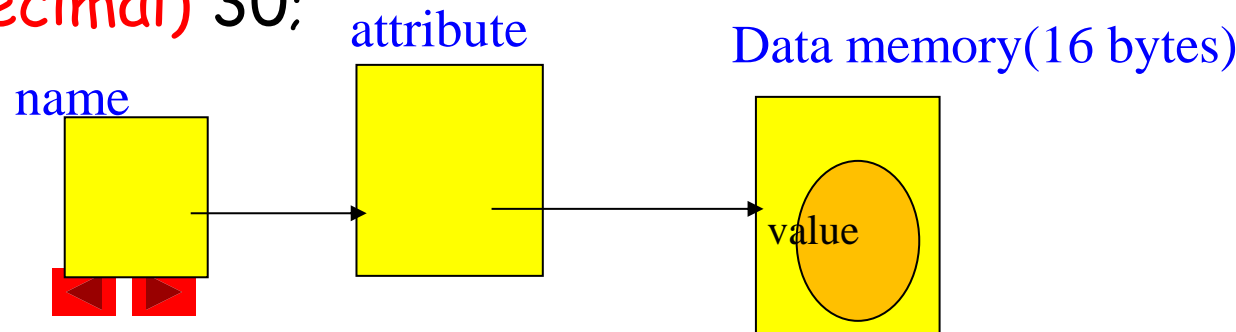
```
19    amount = principal *
20            ( decimal ) Math.Pow( 1.0 + rate, year );
```

- Some Implicit Numeric Conversions
  - Data type conversion from low precision to higher precision is done without indication
  - But not vice versa

From	To
short	int , long, float, double, or decimal
int	long , float, double, or decimal
long	float , double, or decimal
float	double

- Explicit numeric Conversion (called cast)
  - The conversion of data type should be explicitly indicated before the data

Reference Data Types (reference variable)

- Reference data types
  - A variable consisted of several primitive data types and/or the reference data types
    - Ex. String, array, any object variable, ….
  - **Need to "new" the memory for reference variable**
    - **Or, it has referenced to a existing memory**
  - Four boxes for the variable (some complex reference may have more than four boxes)
    - The third box stores the memory of the fourth box
    - The fourth box (the memory) is used store the variable content

name   attribute   Reference address   Data memory

Data's memory address   Data
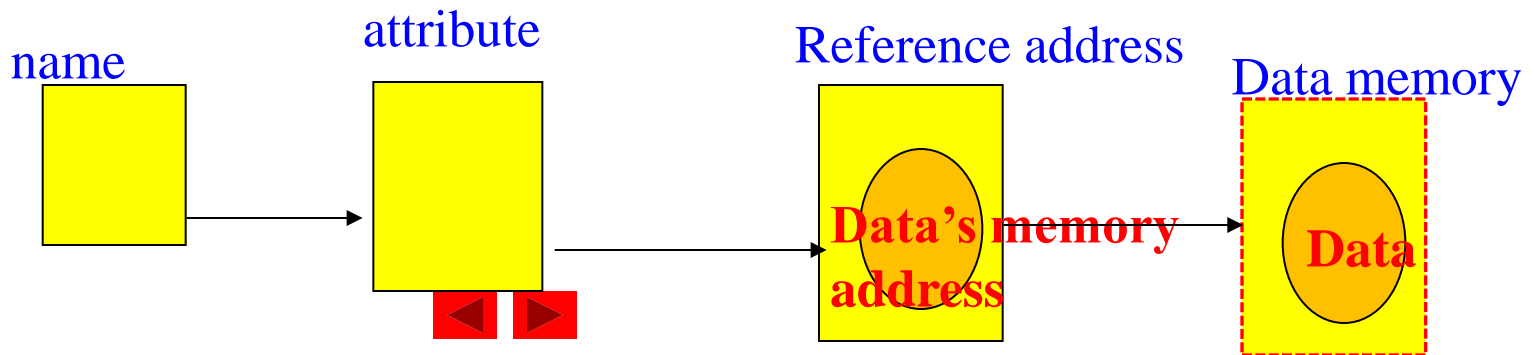
Reference Data Types (reference variable)

- Ex: string st;

  st = "hang";

Memory after declaring reference variable st:

name | attribute | Reference
st → string → ?
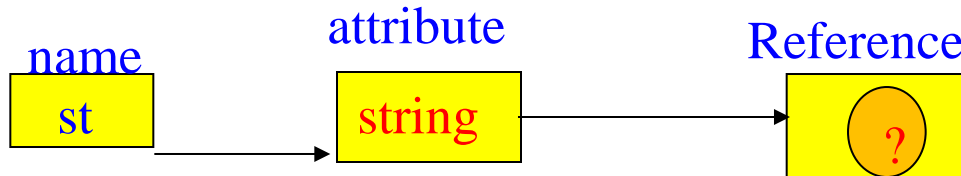
Memory after executing the assignment of st:
(Note: only string is of no need to "new" its memory)
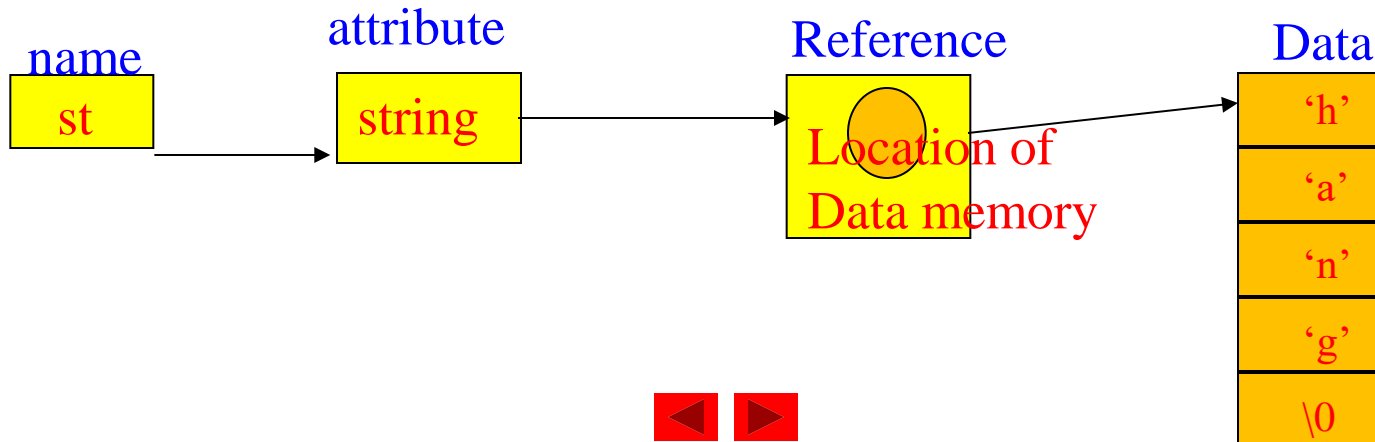
name | attribute | Reference | Data
st → string → Location of Data memory → 'h' 'a' 'n' 'g' \0

5.5 The switch Multiple-Selection Structure

- **switch** structure --Used for multiple selections
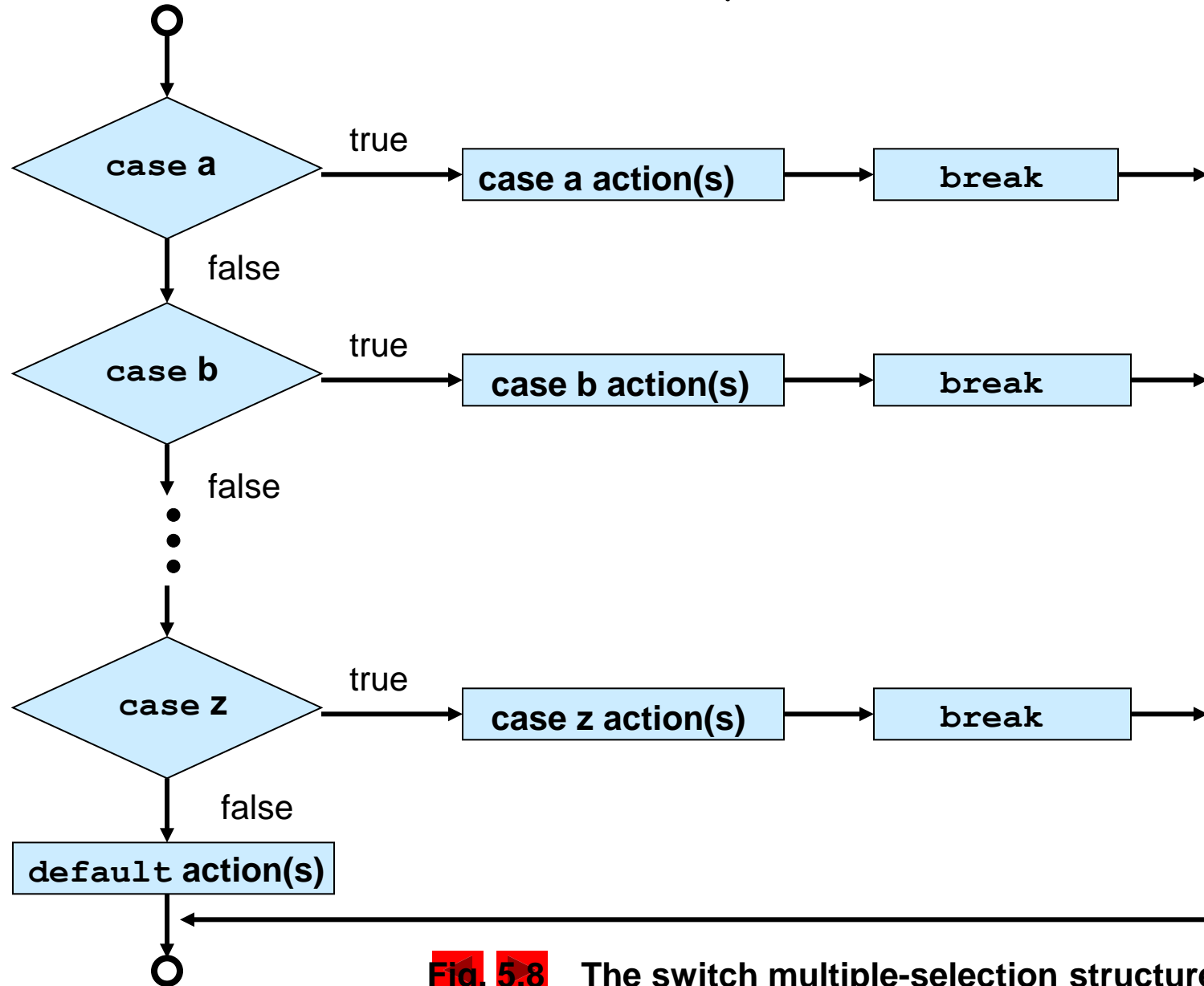
Fig. 5.8   **The switch multiple-selection structure.**

5.5 The switch Multiple-Selection Structure

- Switch is equal to layered "if statement",
- the default is equal to the else of the last "if statement"

int k ;
 if (k == 5)
     k++;
   else if (k ==6)
       k+= 2;
       else if (k == 7)
           k+= 3;
         else
           k+= 4;
	Switch (k) {
	 case 5:
	     k++; **break;**
	 case 6:
	     k+=2; **break;**
	 case 7:
	     k+=3; **break;**
	 default:
	     k+= 4;
	}

5.5 The switch Multiple-Selection Structure

• What if forgetting the break statement in the case?

– The execution will pass to next case statement(s) until

1. encountering the break, or

2. entering the default parts, or

3. encountering the end of switch statement

```
Switch (k) {
case 5:
    k++;
case 6:
    k+=2; break;
case 7:
    k+=3; break;
default:
    k+= 4; }
```
If k = 5, finally k =?

```
Switch (k) {
 case 5:
    k++;
 case 4, 6:
    k+=2;
default:
    k+= 4; }
```
If k = 5, finally k =?

5.5 The switch Multiple-Selection Structure

- **switch** structure --Used for multiple selections
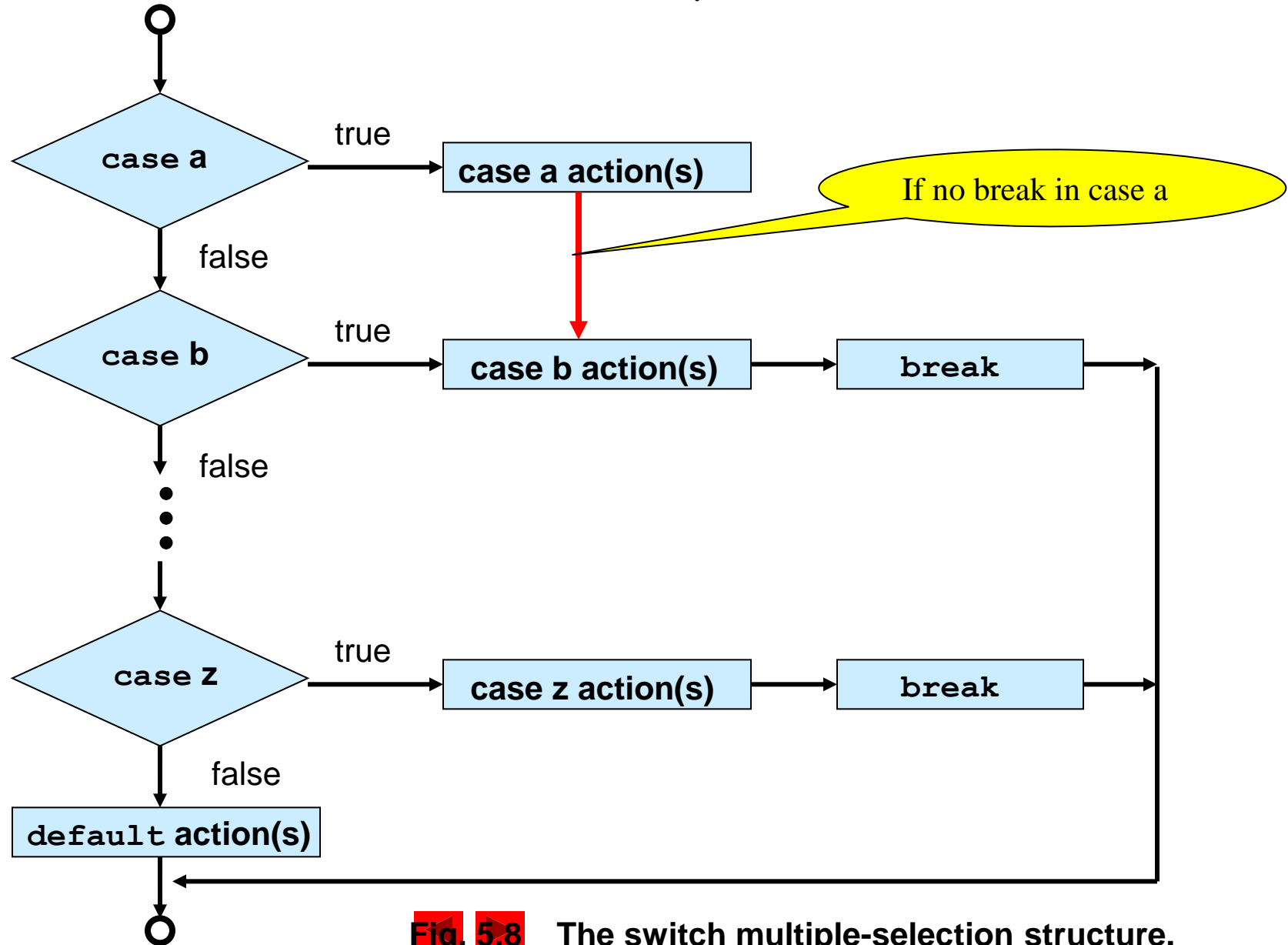
Fig. 5.8 The switch multiple-selection structure.

Example of no break in switch

```
Switch (x){
    case 2:
    system.print("two");
    case 3:
    system.print("Three")
    ; }
```

If x = 2, output is

Two Three

```
switch（month）{
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:

        day=31; break;
    case 4:
    case 6:
    case 9:
    case 11:

        day=30; break;
    case 2:

        day=28; break; }
```

5.5   The switch Multiple-Selection Structure

- In C-like language, testing condition in Switch statement can be any legal conditional express used in if statement
    - Like switch (k+3), switch (k*2-5)

- Eq (not encouraged, since of uncertainty of execution):

        switch (k*2-5) {
            case k+3:
                ……; break;
            case k-4:
                …….
        }

5.5   The switch Multiple-Selection Structure

- in most languages, the case statement use a constant only in its testing condition (to simplify the tracing of the programmer)

- Eq:

```
switch (k) {
   case 'a':
          ......; break;
    case 'b':
          .......
   }
```

5.6   The `do/while` Repetition Structure

- **do/while** structure
  - Similar to **while** structure
  - Tests loop-continuation after performing body of loop
    - i.e., loop body executes at least once
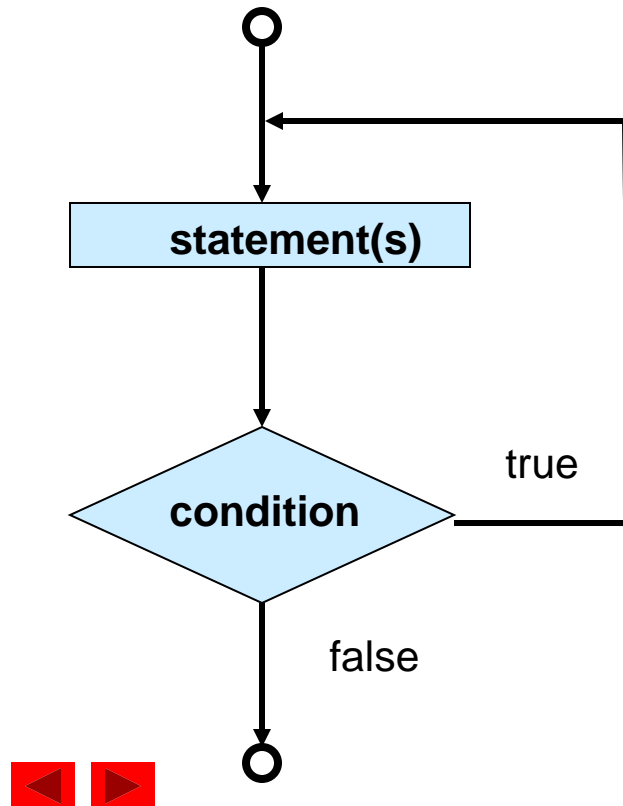
```csharp
using System;

class DoWhileLoop
{
    static void Main( string[] args )
    {
        int counter = 1;

        do
        {
            Console.WriteLine( counter );
            counter++;
        } while ( counter <= 5 );

    } // end method Main

} // end class DoWhileLoop
```

What if '5' is changed to -1?

Output is

1

The output

1
2
3
4
5

Keyword in *C#*

C# Keywords				
abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	get
goto	if	implicit	in	int
interface	internal	is	lock	long
namespace	new	null	object	operator
out	override	params	private	protected
public	readonly	ref	return	sbyte
sealed	set	short	sizeof	stackalloc
static	string	struct	switch	this
throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using
value	virtual	void	volatile	while

Fig. 4.2 C# keywords.

Keyword and reserved words

- In pascal-like language, it uses reserved word to recognize the program structure
  - Ex. for (k=1; k<= 10; k++)

    ......
  - "for" is reserved to represent a loop structure;
  - You cannot use "for" as a variable name
- Keyword:
  - words important as a hint to compiler
    - **class** keyword followed by class name
    - for keyword followed by a loop
- In C-like language (such as C#, Java), it only uses keyword instead of reserved words
  - Keyword provides the growth space for language
  - But keyword provides ambiguity if not appropriate use

Keyword and reserved words

- – Ex. legal codes in *C#* (keyword used)

```
int  if = 1, else = 2, for = 3;
if (if == for)
    for++;
  else {else = for;
          for (for = if; if != else; else++)
              for++; }
```

- Why keyword provides the growth space for language?
  - In pascal language, class is legal variable name;
    - Since no class concept is used in that time;
  - – But in Delphi, class is illegal variable name;
    - Old Pascal program must be revised and recompiled
  - – In contrast, old *C#* or Java program needs not to be revised, even if a new language structure is invented

Basic instruction cycle

1. The CPU fetches the next instruction (with its operands) from memory.
2. Then the CPU executes the instruction.
3. Program counter is automatically incremented after each fetch.

Note: Program counter (PC) holds address of the instruction to be fetched next.

Fetch Cycle	Execute Cycle

START → Fetch Next Instruction → Execute Instruction → HALT
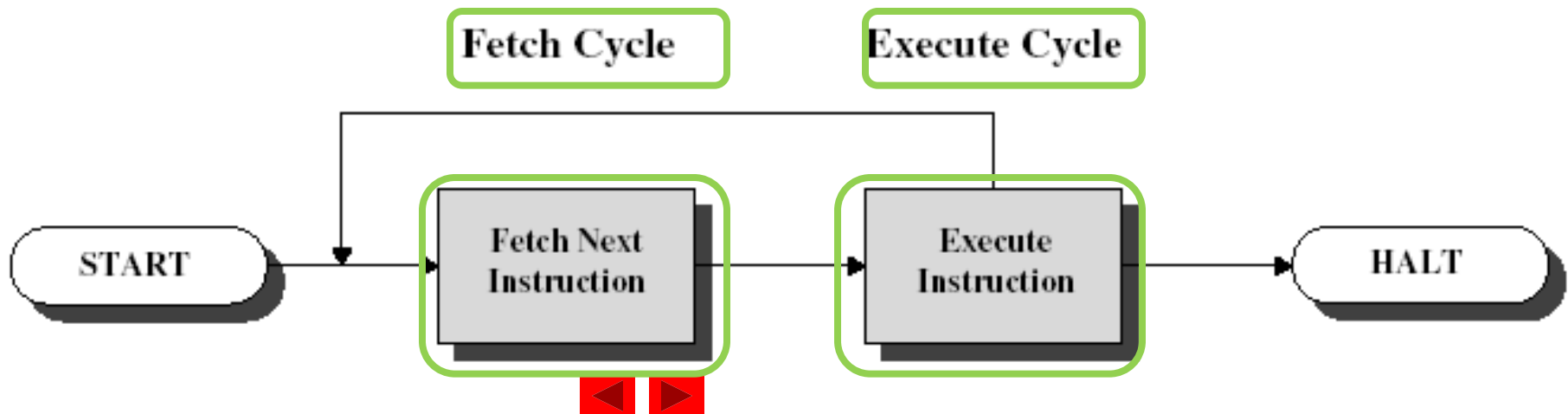
Some Registers in Processor

- Data registers
  - such as AC, BX, …
- Control and status registers
  - Program counter (PC)
    - Contain address of an instruction to be fetched.
  - Instruction register (IR)
    - Contains instruction most recently fetched.
  - ……

Computer components

CPU

PC

IR

Execution unit

System Bus

Main Memory

0
1
2

Instruction
Instruction
Instruction

Data
Data
Data
Data

$n - 2$
$n - 1$

I/O Module

Buffers

PC = Program counter
IR = Instruction register
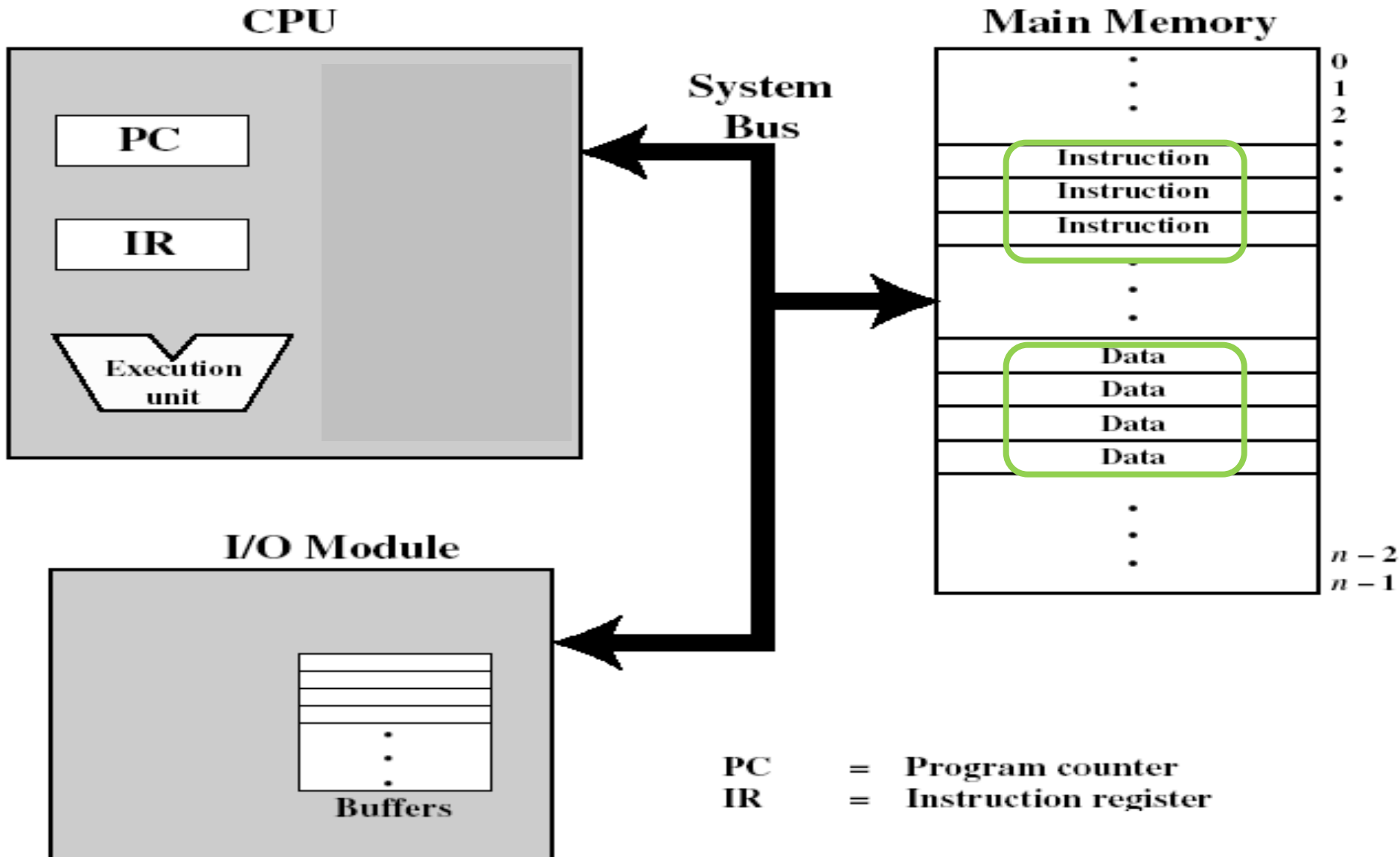
Computation

- Program in C

  .....

  int a = 3, b = 2;

  b = a+b;

  .....

- assumption

  – a is in memory
  address 940

  – b is in memory
  address 941

0001 = Load AC form memory
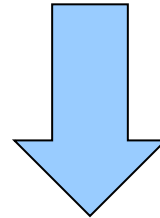0010 = Store AC to memory
0101 = Add to AC from memory
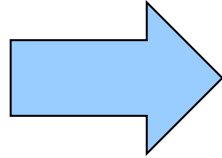
- Assembly codes

  . . .

  load AC, a

  add AC, b

  stor b, AC

- Machine codes

  1 940
  5 941
  2 941

Assume the codes stored from memory 300

load AC, a

add AC, b

stor b, AC

Assume a is in memory address 940

b is in memory address 941

And a = 3; b = 2

Program Counter (PC) = Address of instruction

Instruction Register (IR) = Instruction being executed

Accumulator (AC) = Temporary storage

0001 = Load AC form memory   1

0010 = Store AC to memory   2

0101 = Add to AC from memory   5

Step 1

Memory
300 | 1 9 4 0
301 | 5 9 4 1
302 | 2 9 4 1
940 | 0 0 0 3
941 | 0 0 0 2

CPU Registers
3 0 0 | PC
| AC
1 9 4 0 | IR

Step 2

Memory
300 | 1 9 4 0
301 | 5 9 4 1
302 | 2 9 4 1
940 | 0 0 0 3
941 | 0 0 0 2

CPU Registers
3 0 1 | PC
0 0 0 3 | AC
1 9 4 0 | IR

Step 3

Memory
300 | 1 9 4 0
301 | 5 9 4 1
302 | 2 9 4 1
940 | 0 0 0 3
941 | 0 0 0 2

CPU Registers
3 0 1 | PC
0 0 0 3 | AC
5 9 4 1 | IR

Step 4

Memory
300 | 1 9 4 0
301 | 5 9 4 1
302 | 2 9 4 1
940 | 0 0 0 3
941 | 0 0 0 2

CPU Registers
3 0 2 | PC
0 0 0 5 | AC
5 9 4 1 | IR

$3 + 2 = 5$

Step 5

Memory
300 | 1 9 4 0
301 | 5 9 4 1
302 | 2 9 4 1
940 | 0 0 0 3
941 | 0 0 0 2

CPU Registers
3 0 2 | PC
0 0 0 5 | AC
2 9 4 1 | IR

Step 6

Memory
300 | 1 9 4 0
301 | 5 9 4 1
302 | 2 9 4 1
940 | 0 0 0 3
941 | 0 0 0 5

CPU Registers
3 0 3 | PC
0 0 0 5 | AC
2 9 4 1 | IR