



Computer Programming (a)

E1123

Fall 2022-2023

Lecture 4



Operators and Control Structures

INSTRUCTOR

DR / AYMAN SOLIMAN

➤ Contents

- 1) Operators
- 2) Control Structures
- 3) Conditions
- 4) One-Way Selection
- 5) Two-Way Selection
- 6) Multiple Selections: Nested if
- 7) switch Structures



Operators

```
graph TD; Operators[Operators] --- Arithmetic[Arithmetic]; Operators --- Increment[Increment/decrement]; Operators --- Relational[Relational]; Operators --- Logical[Logical]; Operators --- Bitwise[Bitwise];
```

Arithmetic

**Increment/
decrement**

Relational

Logical

Bitwise

➤ Operators

An **operation** is a mathematical calculation involving zero or more input values (called **operands**) that produces an output value and in mathematics, operators such as + , - , * , / , ...etc.

➤ Precedence and Associativity

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ -- (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

Arithmetic Operators

```
graph TD; A[Arithmetic Operators] --> B[Unary]; A --> C[Binary]
```

Unary

Binary

➤ Arithmetic Operators

➤ Unary

Operator	Symbol	Form	Operation
Unary plus	+	+x	Value of x
Unary minus	-	-x	Negation of x

➤ Binary

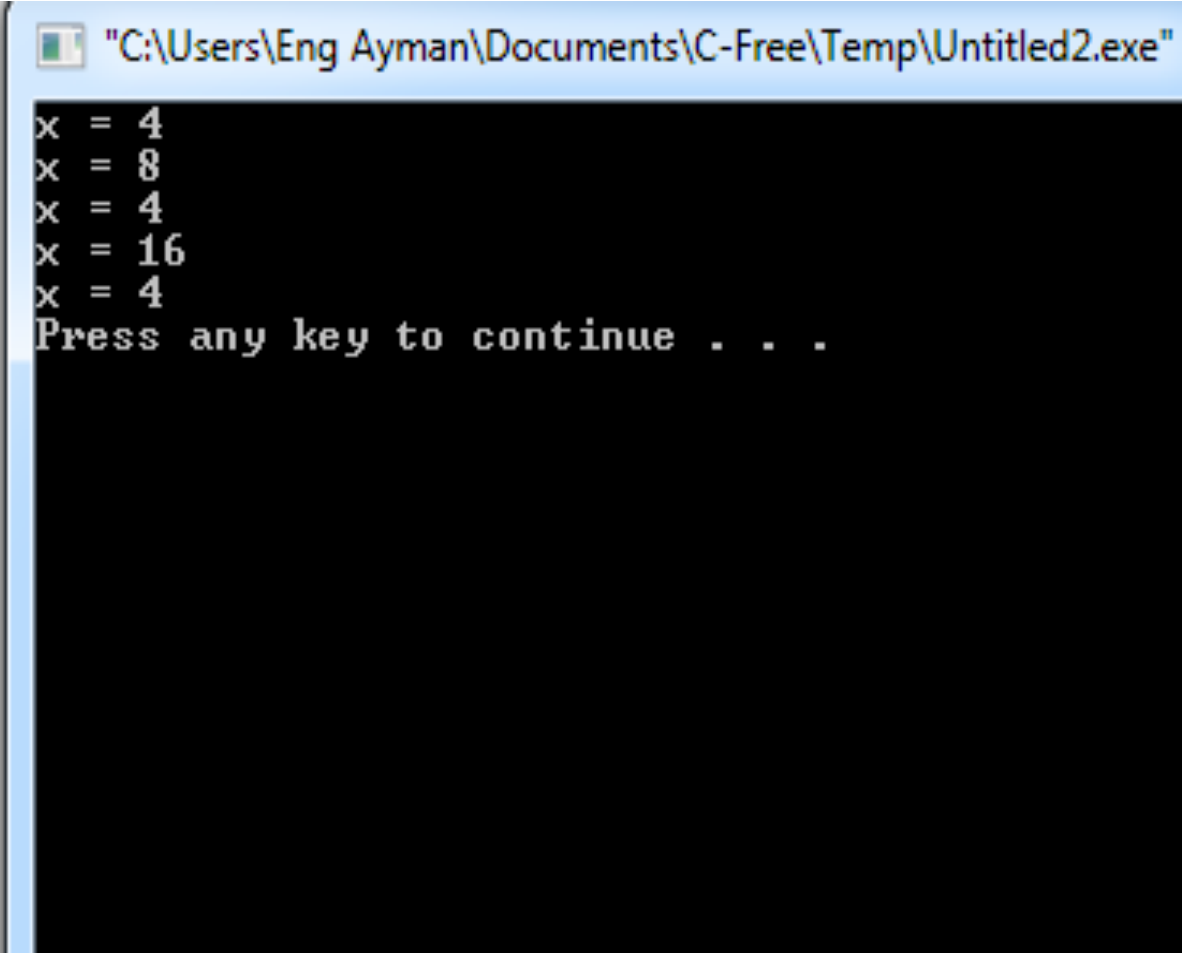
Operator	Symbol	Form	Operation
Addition	+	$x + y$	x plus y
Subtraction	-	$x - y$	x minus y
Multiplication	*	$x * y$	x multiplied by y
Division	/	x / y	x divided by y
Modulus (Remainder)	%	$x \% y$	The remainder of x divided by y

➤ Arithmetic assignment operators

Operator	Symbol	Form	Operation
Assignment	=	$x = y$	Assign value y to x
Addition assignment	+=	$x += y$	Add y to x ($x = x + y$)
Subtraction assignment	-=	$x -= y$	Subtract y from x ($x = x - y$)
Multiplication assignment	*=	$x *= y$	Multiply x by y ($x = x * y$)
Division assignment	/=	$x /= y$	Divide x by y ($x = x / y$)
Modulus assignment	%=	$x \% = y$	Put the remainder of x / y in x ($x = x \% y$)

➤ Example

```
1 #include <iostream.h>
2 int main()
3 {
4     double x=10.5,y=4;
5     x=y;
6     cout<<"x = "<< x <<endl;
7     x += y;
8     cout<<"x = "<< x <<endl;
9     x -= y;
10    cout<<"x = "<< x <<endl;
11    x *= y;
12    cout<<"x = "<< x <<endl;
13    x /= y;
14    cout<<"x = "<< x <<endl;
15    return 0;
16 }
```



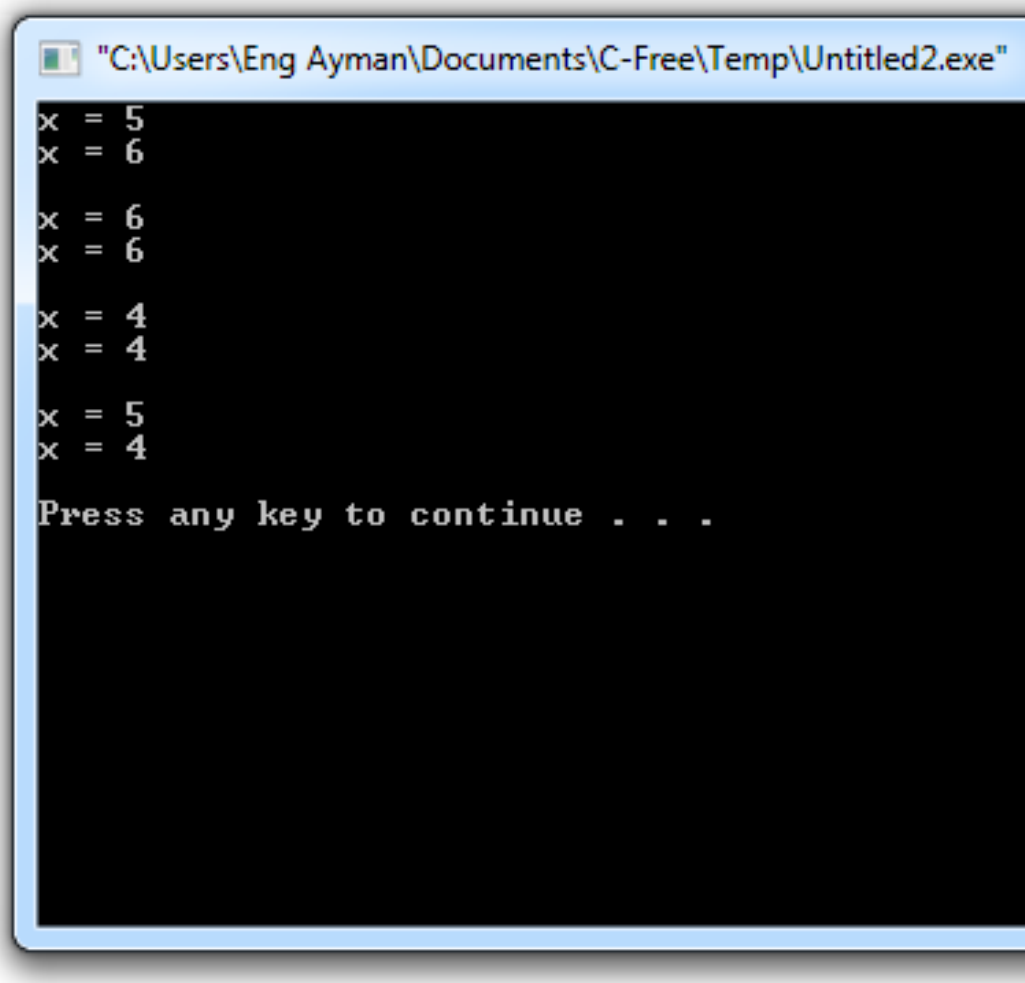
```
"C:\Users\Eng Ayman\Documents\C-Free\Temp\Untitled2.exe"
x = 4
x = 8
x = 4
x = 16
x = 4
Press any key to continue . . .
```


➤ Increment/decrement operators

Operator	Symbol	Form	Operation
Prefix increment (pre-increment)	++	++x	Increment x, then evaluate x
Prefix decrement (pre-decrement)	--	--x	Decrement x, then evaluate x
Postfix increment (post-increment)	++	x++	Evaluate x, then increment x
Postfix decrement (post-decrement)	--	x--	Evaluate x, then decrement x

➤ Example

```
1 #include <iostream.h>
2 int main()
3 {
4     double x=5;
5     cout<<"x = "<< x++ <<endl;
6     cout<<"x = "<< x <<endl;
7     cout<<endl;
8     x=5;
9     cout<<"x = "<< ++x <<endl;
10    cout<<"x = "<< x <<endl;
11    cout<<endl;
12    x=5;
13    cout<<"x = "<< --x <<endl;
14    cout<<"x = "<< x <<endl;
15    cout<<endl;
16    x=5;
17    cout<<"x = "<< x-- <<endl;
18    cout<<"x = "<< x <<endl;
19    cout<<endl;
20    return 0;
21 }
```



```
"C:\Users\Eng Ayman\Documents\C-Free\Temp\Untitled2.exe"
x = 5
x = 6

x = 6
x = 6

x = 4
x = 4

x = 5
x = 4

Press any key to continue . . .
```

➤ Relational Operators (Comparisons)

Operator	Symbol	Form	Operation
Greater than	>	$x > y$	true if x is greater than y, false otherwise
Less than	<	$x < y$	true if x is less than y, false otherwise
Greater than or equals	>=	$x >= y$	true if x is greater than or equal to y, false otherwise
Less than or equals	<=	$x <= y$	true if x is less than or equal to y, false otherwise
Equality	==	$x == y$	true if x equals y, false otherwise
Inequality	!=	$x != y$	true if x does not equal y, false otherwise

➤ Logical operators

Operator	Symbol	Form	Operation
Logical NOT	!	!x	true if x is false, or false if x is true
Logical AND	&&	x && y	true if both x and y are true, false otherwise
Logical OR		x y	true if either x or y are true, false otherwise

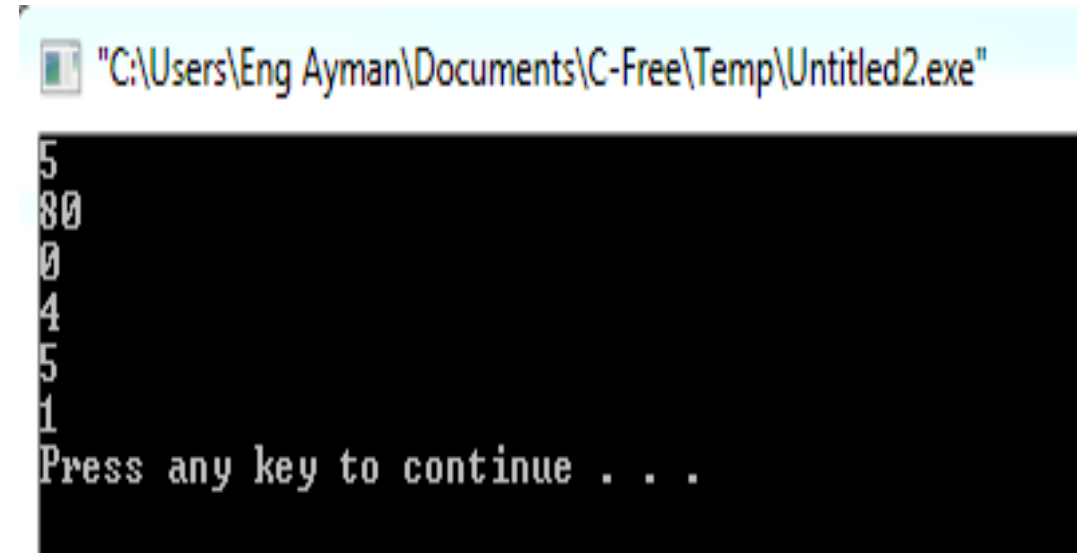
➤ Bitwise Operators

Using bitwise operators, it is possible to write functions that allow us to compact 8 booleans into a single byte-sized variable, enabling significant memory savings at the expense of more complex code.

Operator	Symbol	Form	Operation
left shift	<<	$x \ll y$	all bits in x shifted left y bits
right shift	>>	$x \gg y$	all bits in x shifted right y bits
bitwise NOT	~	$\sim x$	all bits in x flipped
bitwise AND	&	$x \& y$	each bit in x AND each bit in y
bitwise OR		$x y$	each bit in x OR each bit in y
bitwise XOR	^	$x \wedge y$	each bit in x XOR each bit in y

➤ Example

```
#include <iostream.h>
int main()
{
int x = 5, y = 4, z;
cout << x << '\n'; //x= 00000000 00000000 00000000 00000101 = 5
z = x << y;
cout << z << '\n'; //0101 << 4 = 01010000 = 80
z = x >> y;
cout << z << '\n'; //0101 >> 4 = 0000 = zero
z = x & y;
cout << z << '\n'; //0101 & 0100 = 0100 =4
z = x | y;
cout << z << '\n'; //0101 | 0100= 0101 =5
z = x ^ y;
cout << z << '\n'; //0101^0100= 0001 =1
return 0;
}
```



```
"C:\Users\Eng Ayman\Documents\C-Free\Temp\Untitled2.exe"
5
80
0
4
5
1
Press any key to continue . . .
```

Control Structures

```
graph TD; A[Control Structures] --> B[In sequence]; A --> C[Selectively]; A --> D[Repetitively];
```

In sequence

Selectively

Repetitively

➤ Control Structures

➤ A computer can proceed:

- ❑ In sequence

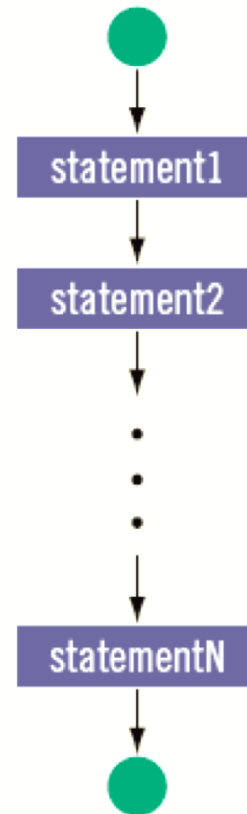
- ❑ Selectively (branch) - making a choice

- ❑ Repetitively (iteratively) – looping

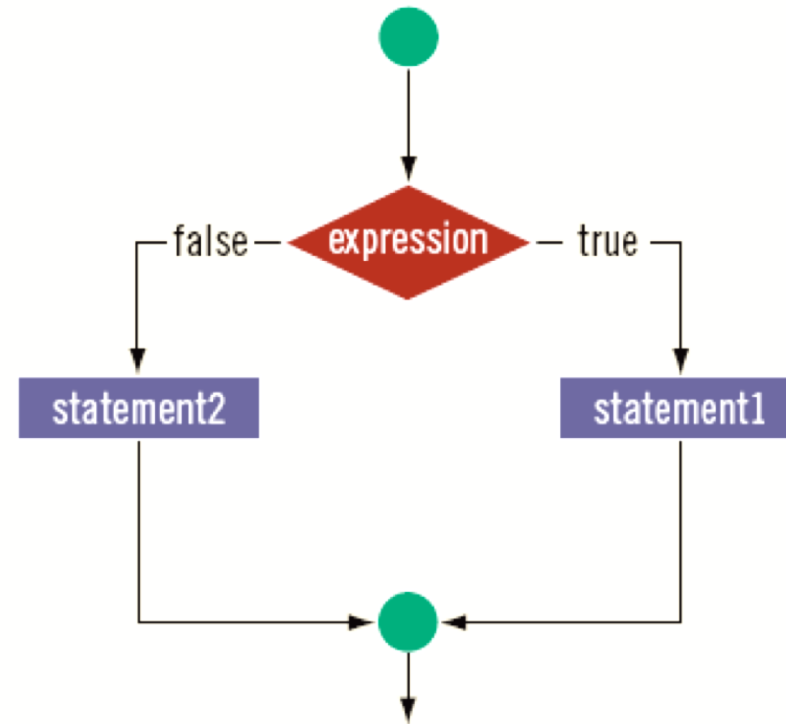
➤ Some statements are **executed only** if certain conditions are met

➤ A condition is met if it evaluates to **true**

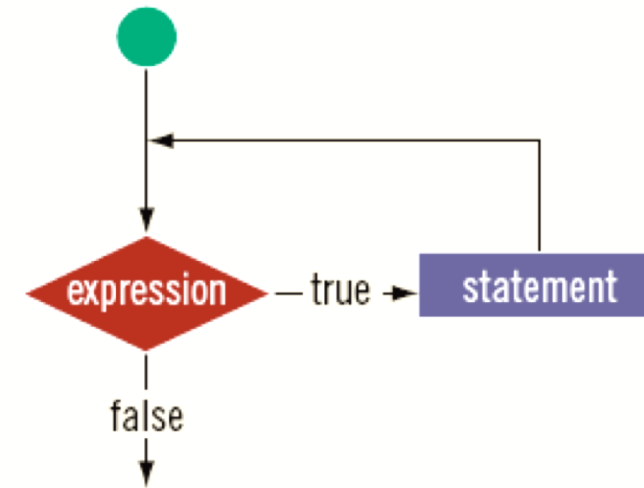
➤ Control Structures (cont.)



a. Sequence



b. Selection



c. Repetition

➤ Relational Operators and Simple Data Types

You can use the relational operators with all three simple data types:

In the following example, the expressions use both integers and real numbers:

$8 < 15$ evaluates to \rightarrow **true**

$6 \neq 6$ evaluates to \rightarrow **false**

$2.5 > 5.8$ evaluates to \rightarrow **false**

$5.9 \leq 7.5$ evaluates to \rightarrow **true**

➤ Comparing Characters

ASCII Value	Char	ASCII Value	Char	ASCII Value	Char	ASCII Value	Char
32	' '	61	=	81	Q	105	i
33	!	62	>	82	R	106	j
34	"	65	A	83	S	107	k
42	*	66	B	84	T	108	l
43	+	67	C	85	U	109	m
45	-	68	D	86	V	110	n
47	/	69	E	87	W	111	o
48	0	70	F	88	X	112	p
49	1	71	G	89	Y	113	q
50	2	72	H	90	Z	114	r
51	3	73	I	97	a	115	s
52	4	74	J	98	b	116	t
53	5	75	K	99	c	117	u
54	6	76	L	100	d	118	v
55	7	77	M	101	e	119	w
56	8	78	N	102	f	120	x
57	9	79	O	103	g	121	y
60	<	80	P	104	h	122	z

Expression	Value of Expression	Explanation
' ' < 'a'	true	The ASCII value of ' ' is 32, and the ASCII value of 'a' is 97. Because 32 < 97 is true, it follows that ' ' < 'a' is true.
'R' > 'T'	false	The ASCII value of 'R' is 82, and the ASCII value of 'T' is 84. Because 82 > 84 is false, it follows that 'R' > 'T' is false.
'+' < '*'	false	The ASCII value of '+' is 43, and the ASCII value of '*' is 42. Because 43 < 42 is false, it follows that '+' < '*' is false.
'6' <= '>'	true	The ASCII value of '6' is 54, and the ASCII value of '>' is 62. Because 54 <= 62 is true, it follows that '6' <= '>' is true.

➤ Relational Operators and the string Type

- Relational operators can be applied to **strings**
- Strings are compared **character by character**, starting with the first character
- Comparison **continues until either a mismatch is found**, or all characters are found equal
- If two strings of different lengths are compared and the comparison is equal to the last character of the shorter string
- The shorter string is less than the larger string

➤ Example

Suppose we have the following declarations:

```
string str1 = "Hello";
```

```
string str2 = "Hi";
```

```
string str3 = "Air";
```

```
string str4 = "Bill";
```

```
string str4 = "Big";
```

Expression	Value	Explanation
<code>str1 < str2</code>	<code>true</code>	<code>str1 = "Hello"</code> and <code>str2 = "Hi"</code> . The first characters of <code>str1</code> and <code>str2</code> are the same, but the second character 'e' of <code>str1</code> is less than the second character 'i' of <code>str2</code> . Therefore, <code>str1 < str2</code> is <code>true</code> .
<code>str1 > "Hen"</code>	<code>false</code>	<code>str1 = "Hello"</code> . The first two characters of <code>str1</code> and "Hen" are the same, but the third character 'l' of <code>str1</code> is less than the third character 'n' of "Hen". Therefore, <code>str1 > "Hen"</code> is <code>false</code> .
<code>str3 < "An"</code>	<code>true</code>	<code>str3 = "Air"</code> . The first characters of <code>str3</code> and "An" are the same, but the second character 'i' of "Air" is less than the second character 'n' of "An". Therefore, <code>str3 < "An"</code> is <code>true</code> .

➤ Example

Expression	Value	Explanation
<code>str4 >= "Billy"</code>	<code>false</code>	<code>str4 = "Bill"</code> . It has four characters and "Billy" has five characters. Therefore, <code>str4</code> is the shorter string. All four characters of <code>str4</code> are the same as the corresponding first four characters of "Billy", and "Billy" is the larger string. Therefore, <code>str4 >= "Billy"</code> is <code>false</code> .
<code>str5 <= "Bigger"</code>	<code>true</code>	<code>str5 = "Big"</code> . It has three characters and "Bigger" has six characters. Therefore, <code>str5</code> is the shorter string. All three characters of <code>str5</code> are the same as the corresponding first three characters of "Bigger", and "Bigger" is the larger string. Therefore, <code>str5 <= "Bigger"</code> is <code>true</code> .

Expression	Value	Explanation
<code>str1 == "hello"</code>	<code>false</code>	<code>str1 = "Hello"</code> . The first character 'H' of <code>str1</code> is less than the first character 'h' of "hello" because the ASCII value of 'H' is 72, and the ASCII value of 'h' is 104. Therefore, <code>str1 == "hello"</code> is <code>false</code> .
<code>str3 <= str4</code>	<code>true</code>	<code>str3 = "Air"</code> and <code>str4 = "Bill"</code> . The first character 'A' of <code>str3</code> is less than the first character 'B' of <code>str4</code> . Therefore, <code>str3 <= str4</code> is <code>true</code> .
<code>str2 > str4</code>	<code>true</code>	<code>str2 = "Hi"</code> and <code>str4 = "Bill"</code> . The first character 'H' of <code>str2</code> is greater than the first character 'B' of <code>str4</code> . Therefore, <code>str2 > str4</code> is <code>true</code> .

Conditions

```
graph TD; A[Conditions] --- B[One-Way]; A --- C[Two-Way]; A --- D[Multiple - Nested];
```

One-Way

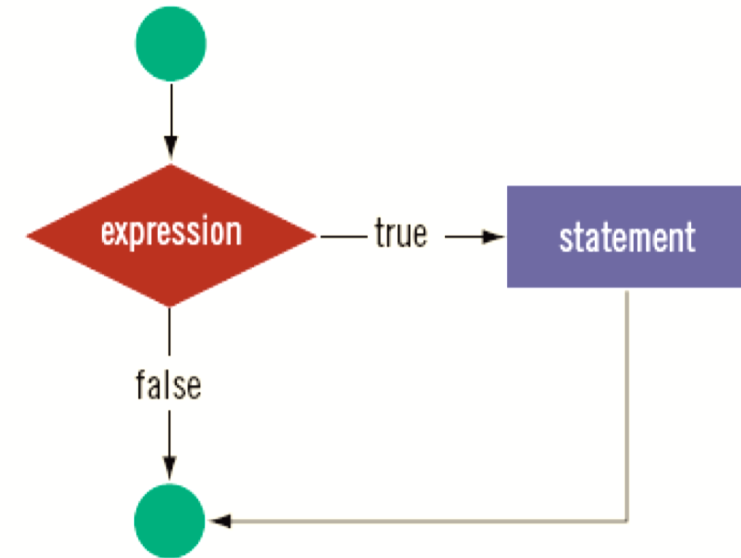
Two-Way

Multiple - Nested

➤ One-Way Selection

The syntax of one-way selection is:

```
if ( expression )  
    statement
```



- ❑ The statement is executed if the value of the expression is **true**
- ❑ The statement is bypassed if the value is **false**; program goes to the next statement
- ❑ **if** is a reserved word

➤ One-Way Selection (syntax error)

Consider the following statement:

```
if score >= 60      //syntax error
  grade = 'P';
```

This statement illustrates an incorrect version of an **if** statement. The parentheses around the logical expression are missing, which is a syntax error.

Consider the following C++ statements:

```
if (score >= 60);      //Line 1
  grade = 'P';         //Line 2
```

Because there is a semicolon at the end of the expression (see Line 1), the **if** statement in Line 1 terminates. The action of this **if** statement is null, and the statement in Line 2 is not part of the **if** statement in Line 1. Hence, the statement in Line 2 executes regardless of how the **if** statement evaluates.

➤ Example

The following C++ program finds the absolute value of an integer:

```
//Program: Absolute value of an integer

#include <iostream>

using namespace std;

int main()
{
    int number, temp;

    cout << "Line 1: Enter an integer: ";           //Line 1
    cin >> number;                                  //Line 2
    cout << endl;                                   //Line 3

    temp = number;                                  //Line 4

    if (number < 0)                                 //Line 5
        number = -number;                           //Line 6

    cout << "Line 7: The absolute value of "
         << temp << " is " << number << endl;      //Line 7

    return 0;
}
```

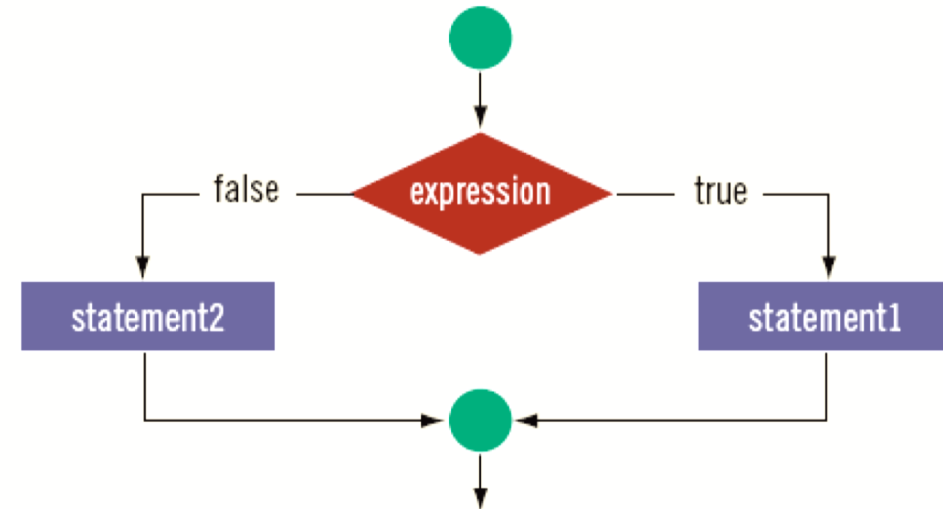
Sample Run: In this sample run, the user input is shaded.

```
Line 1: Enter an integer: -6734
Line 7: The absolute value of -6734 is 6734
```

➤ Two-Way 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

Example → Consider the following statements:

```
if (hours > 40.0) //Line 1  
    wages = 40.0 * rate +  
            1.5 * rate * (hours - 40.0); //Line 2  
else //Line 3  
    wages = hours * rate; //Line 4
```

➤ **Compound (Block of) Statement**

Compound statement (block of statements):

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

➤ Multiple Selections: 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**

Suppose that `balance` and `interestRate` are variables of type `double`. The following statements determine the `interestRate` depending on the value of the `balance`:

```
if (balance > 50000.00)           //Line 1
    interestRate = 0.07;         //Line 2
else                               //Line 3
    if (balance >= 25000.00)     //Line 4
        interestRate = 0.05;    //Line 5
    else                           //Line 6
        if (balance >= 1000.00) //Line 7
            interestRate = 0.03; //Line 8
        else                       //Line 9
            interestRate = 0.00; //Line 10
```

most efficient method

```
if (balance > 50000.00)
    interestRate = 0.07;
else if (balance >= 25000.00)
    interestRate = 0.05;
else if (balance >= 1000.00)
    interestRate = 0.03;
else
    interestRate = 0.00;
```

➤ Example

Assume that `score` is a variable of type `int`. based on the value of `score`, the following code outputs the grade:

```
if (score >= 90)
    cout << "The grade is A." << endl;
else if (score >= 80)
    cout << "The grade is B." << endl;
else if (score >= 70)
    cout << "The grade is C." << endl;
else if (score >= 60)
    cout << "The grade is D." << endl;
else
    cout << "The grade is F." << endl;
```

➤ Comparing if...else Statements with a Series of if Statements

First method

```
if (month == 1)
    cout << "January" << endl;
if (month == 2)
    cout << "February" << endl;
if (month == 3)
    cout << "March" << endl;
if (month == 4)
    cout << "April" << endl;
if (month == 5)
    cout << "May" << endl;
if (month == 6)
    cout << "June" << endl;
```

Second method

```
if (month == 1)
    cout << "January" << endl;
else if (month == 2)
    cout << "February" << endl;
else if (month == 3)
    cout << "March" << endl;
else if (month == 4)
    cout << "April" << endl;
else if (month == 5)
    cout << "May" << endl;
else if (month == 6)
    cout << "June" << endl;
```



➤ Which method is preferred?

➤ Associativity of Relational Operators:

```
#include <iostream>

using namespace std;

int main()
{
    int num;

    cout << "Enter an integer: ";
    cin >> num;
    cout << endl;

    if (0 <= num <= 10)
        cout << num << " is within 0 and 10." << endl;
    else
        cout << num << " is not within 0 and 10." << endl;

    return 0;
}
```

Solution:

Sample Runs:

Sample Run 1:

Enter an integer: 5

5 is within 0 and 10. (correct)

Sample Run 2:

Enter an integer: 20

20 is within 0 and 10. (incorrect)

Sample Run 3:

Enter an integer: -10

-10 is within 0 and 10. (incorrect)



<code>0 <= num <= 10</code>	<code>= 0 <= 5 <= 10</code>	
	<code>= (0 <= 5) <= 10</code>	(Because relational operators are evaluated from left to right)
	<code>= 1 <= 10</code>	(Because <code>0 <= 5</code> is true , <code>0 <= 5</code> evaluates to 1)
	<code>= 1 (true)</code>	

Now, suppose that `num = 20`. Then:

<code>0 <= num <= 10</code>	<code>= 0 <= 20 <= 10</code>	
	<code>= (0 <= 20) <= 10</code>	(Because relational operators are evaluated from left to right)
	<code>= 1 <= 10</code>	(Because <code>0 <= 20</code> is true , <code>0 <= 20</code> evaluates to 1)
	<code>= 1 (true)</code>	

`(0 <= num && num <= 10)`

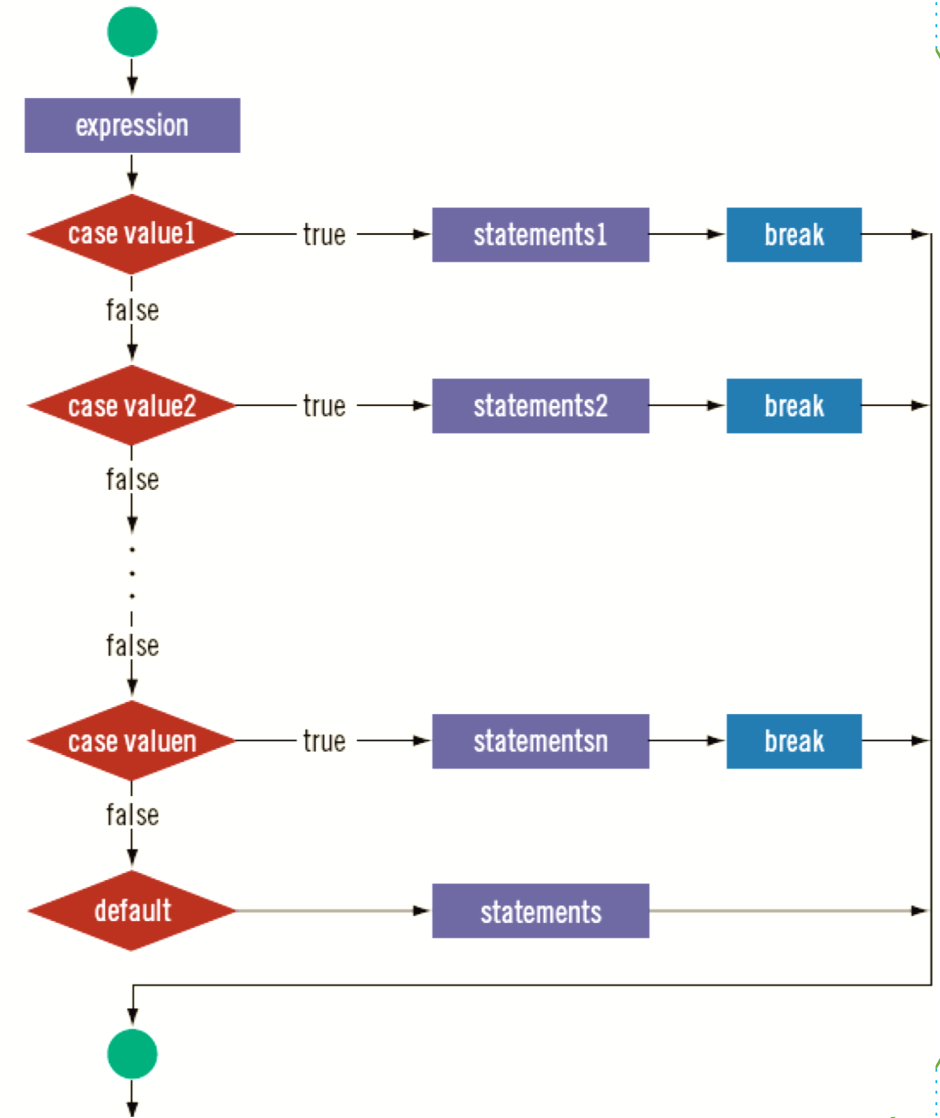
➤ switch Structures

- ❑ switch structure: alternate to if-else
- ❑ switch (integral) expression is evaluated first
- ❑ Value of the **expression** determines which **corresponding action** is taken
- ❑ **Expression** is sometimes called the **selector**

```
switch (expression)
{
  case value1:
    statements1
    break;
  case value2:
    statements2
    break;
  .
  .
  .
  case valuen:
    statementsn
    break;
  default:
    statements
}
```

➤ switch Structures (cont.)

- ❑ 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



➤ Example

Consider the following statements, where `grade` is a variable of type `char`:

```
switch (grade)
{
case 'A':
    cout << "The grade is 4.0.";
    break;
case 'B':
    cout << "The grade is 3.0.";
    break;
case 'C':
    cout << "The grade is 2.0.";
    break;
case 'D':
    cout << "The grade is 1.0.";
    break;
case 'F':
    cout << "The grade is 0.0.";
    break;
default:
    cout << "The grade is invalid.";
}
```

In this example, the expression in the `switch` statement is a variable identifier. The variable `grade` is of type `char`, which is an integral type. The possible values of `grade` are 'A', 'B', 'C', 'D', and 'F'. Each `case` label specifies a different action to take, depending on the value of `grade`. If the value of `grade` is 'A', the output is:

The grade is 4.0.

➤ Example (attention)

```
int main()
{
    int num;


    cout << "Enter an integer between 0 and 7: ";

    cin >> num;

    switch(num)
    {
        case 0:
        case 1:
            cout << "Learning to use ";
        case 2:
            cout << "C++'s ";
        case 3:
            cout << "switch structure." << endl;
            break;
        case 4:
            break;
        case 5:
            cout << "This program shows the effect ";
        case 6:
        case 7:
            cout << "of the break statement." << endl;
            break;
        default:
            cout << "The number is out of range." << endl;
    }

    cout << "Out of the switch structure." << endl;

    return 0;
}
```

 "C:\Users\Eng Ayman\Documents\C-Free\Temp\Untitled2.exe"

```
Enter an integer between 0 and 7: 5
This program shows the effect of the break statement.
Out of the switch structure.
Press any key to continue . . .
```

*Thank
you*

