CHAPTER 1.3
THE OPERATORS

Dr. Shady Yehia Elmashad

Outline

1. Arithmetic Operators
2. Accumulation Operators
3. Incremental/ Decremental Operators
4. Equality/Relational Operators
5. Logical Operators
6. Confusing Equality (==) and Assignment (=) Operators

1. Arithmetic Operators

Operator	Symbol	Action	Example
Addition	+	Adds operands	x + y
Subtraction	-	Subs second from first	x - y
Negation	-	Negates operand	-x
Multiplication	*	Multiplies operands	x * y
Division	/	Divides first by second (integer quotient)	x / y
Modulus	%	Remainder of divide op	x % y

1. Arithmetic Operators

Example

```
float   a = 31/3;
```
a = 10.3

```
float   b = 31%3;
```
b = 1.00

```
int     c = 31/3;
```
c = 10

```
int     d = 31%3;
```
d = 1

contoso

1. Arithmetic Operators

Example: What is the output?

```
#include<iostream.h>
void main()
{    float  sum = 0 ;
cout<< " the value of sum is initially set to "  <<
sum<<endl;
sum = sum + 98 ;
cout<<"sum is now: " << sum << endl ;
sum = sum – 70 ;
cout<<" sum is now: " << sum<< endl ;
sum = sum * 20 ;
cout<<"sum is now : " <<sum<<endl;
sum= sum / 6 ;
cout<<"sum is now:"<<sum<<endl;
sum=sum%3 ;
cout<<"sum is now:"<<sum<<endl;
}
```

1. Arithmetic Operators

Operator precedence

- Some arithmetic operators act before others (i.e., multiplication before addition)
  - Be sure to use parenthesis when needed

- Example:

  Find the average of three variables a, b and c
  - Do not use: `a + b + c / 3`
  - Use: `(a + b + c ) / 3`

1. Arithmetic Operators

Operator precedence

- Rules of operator precedence:

Operator(s)	Operation(s)	Order of evaluation (precedence)
`()`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`*`, `/`, or `%`	Multiplication Division Modulus	Evaluated second. If there are several, they are evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

1. Arithmetic Operators

Example: What is the output?

```
#include<iostream.h>
void main(  )
{
        float    a, b, c, d ;
        a = 8 + 2 * 3 ;
        b = ( 5 * 2 – 3 ) / 6;
        c = 5 * 2 – 3 / 6;
        d = 4 + 2 / 4 * 8;
cout << “a=”<< a<<endl << “b=”<<
b<<endl;
cout << “c=”<< c<<endl << “d=”<<
d<<endl;
}
```

1. Arithmetic Operators

Example: Calculate the average of three numbers

```
#include<iostream.h>
void main(  )
{
   float  avg, grade1, grade2, grade3 ;
   grade1 = 8.5;   grade2 = 12.0 ;   grade3 = 9.0;
   avg = grade1 + grade2 + grade3 / 3.0;
cout<<”the average  is”
<<setprecision(1)<<avg;
}
```

```
avg = ( grade1 + grade2 + grade3 )/3.0 ;
```

2. Accumulation/Assignment Operators

- Assignment expression abbreviations

  `c = c + 3;` can be abbreviated as `c += 3;`

  using the addition assignment operator

- Statements of the form

```
variable = variable operator expression;
```

  can be rewritten as

```
variable operator= expression;
```

2. Accumulation/Assignment Operators

Operator	Expression	Alternative
+ =	sum = sum + 10 ;	sum += 10 ;
- =	score = score – 22 ;	score – = 22 ;
* =	x = x * z;	x *= z;
/ =	x = x / y;	x /= y;
% =	x = x % y;	x %= y;

3. Incremental/ Decremental Operators

Operator	Expression	Alternative
Incremental	i = i +1	i++ Or ++i
Decremental	i = i - 1	i-- Or --i

3. Incremental/ Decremental Operators

- Preincrement
  - When the operator is used before the variable (**`++c`** or **`--c`**)
  - Variable is changed, then the expression it is in is evaluated.
- Posincrement
  - When the operator is used after the variable (**`c++`** or **`c--`**)
  - Expression the variable is in executes, then the variable is changed.

- Example:

  If **`c = 5`**, then

- **`cout << ++c;`** prints out **6** (**`c`** is changed before **`cout`** is executed)
- **`cout << c++;`** prints out **5** (**`cout`** is executed before the increment. **`c`** now has the value of **6**)

3. Incremental/ Decremental Operators

- When Variable is not in an expression
  - Preincrementing and postincrementing have the same effect.

```
++c;
cout << c;
```

and

```
c++;
cout << c;
```

have the same effect.

4. Equality/Relational Operators

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	>	`x > y`	**x** is greater than **y**
<	<	`x < y`	**x** is less than **y**
≥	>=	`x >= y`	**x** is greater than or equal to **y**
≤	<=	`x <= y`	**x** is less than or equal to **y**
Equality operators			
=	==	`x == y`	**x** is equal to **y**
≠	!=	`x != y`	**x** is not equal to **y**

```
1  // Fig. 1.14: fig01_14.cpp
2  // Using if statements, relational
3  // operators, and equality operators
4  #include <iostream>
5
6  using std::cout;  // program uses cout
7  using std::cin;   // program uses cin
8  using std::endl;  // program uses endl
9
10 int main()
11 {
12    int num1, num2;
13
14    cout << "Enter two integers, and I will tell you\n"
15         << "the relationships they satisfy: ";
16    cin >> num1 >> num2;   // read two integers
17
18    if ( num1 == num2 )
19       cout << num1 << " is equal to " << num2 << endl;
20
21    if ( num1 != num2 )
22       cout << num1 << " is not equal to " << num2 << endl;
23
24    if ( num1 < num2 )
25       cout << num1 << " is less than " << num2 << endl;
26
27    if ( num1 > num2 )
28       cout << num1 << " is greater than " << num2 << endl;
29
30    if ( num1 <= num2 )
31       cout << num1 << " is less than or equal to "
32            << num2 << endl;
33
```

Notice the **using** statements.

```
Enter two integers, and I will tell you
the relationships they satisfy:
3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal
```

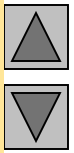
The **if** statements test the truth of the condition. If it is true, the body of the **if** statement is executed. If not, body is skipped.

To include multiple statements in a body, delineate them with braces **{ }**.

Outline

1. Load **<iostream>**
2. **main**
   - 2.1 Initialize **num1** and **num2**
   - 2.1.1 Input data
   - **tatements**

```
34    if ( num1 >= num2 )
35       cout << num1 << " is greater than or
36            << num2 << endl;
37
38    return 0;    // indicate that program ended
39}
```



Outline

2.3 exit (`return 0`)

Program Output

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7

Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12

Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

5. Logical Operators

Operator	Meaning	Example
&&	AND	If(x > y && x<= 20)
||	OR	If(x>y || x< 30)
!	NOT	If(! x)

5. Logical Operators

- **`&&`** (logical **`AND`**)
  - Returns **`true`** if both conditions are **`true`**

- **`||`** (logical **`OR`**)
  - Returns **`true`** if either of its conditions are **`true`**

- **`!`** (logical **`NOT`**, logical negation)
  - Reverses the truth/falsity of its condition
  - Returns **`true`** when its condition is **`false`**

  I- s a unary operator, only takes one condition

- Logical operators used as conditions in loops

5. Logical Operators

Truth Tables

AND Gate

A	B	A &&B
T	T	T
T	F	F
F	T	F
F	F	F

OR Gate

A	B	A || B
T	T	T
T	F	T
F	T	T
F	F	F

NOT Gate

A	! A
T	F
F	T

5. Logical Operators

Example

- **Given `int i=3, k=5, j=0, m=-2;`**
- **Evaluate:**
  - `(0 < i)  &&  (i < 5)`
  - `(i > k)  ||  (j < i)`
  - `! (k > 0)`
  - `i+j < k`
  - `(i < 0)  &&  (j < 7)`
  - `(i < k)  ||  (j < 7)`
  - `(m > k)  ||  (j > 0)`
  - `3*i - 4/k < 2`

5. Logical Operators

Example: What is the output?

- **Given `int i=4;`**
- **Evaluate:**

```
cout << (14+4*4 < 5*(4+3)- ++i);
          14+16  < 5*7 - ++i
          30     < 35  - 5
          30     < 30
```

```
cout << (14+4*4  > 5*(4+3) - i++ -1)
          14+16   > 5*7  - i++ - 1
          30      > 35   - 4   - 1
          30      > 30
```

5. Logical Operators

Short Circuiting

- C++ is very economical when evaluating Boolean expression.
- Therefore, if in the evaluation of a compound Boolean expression, the computer can determine the value of the whole expression without any further evaluation, it does so. This called short circuiting.

- (True || expression) ---------- True
- (False && expression) ---------- False

Example:
Given: int A = 17, B = 65, C = 21, D = 19;

(13 < = A) || (A < = 19)
(D > = C) && (B > = C)
! (C < = B) && ! (D < = C)

contoso

6. Confusing Equality (==) and Assignment (=) Operators

- These errors are damaging because they do not ordinarily cause syntax errors.
  - Recall that any expression that produces a value can be used in control structures. Nonzero values are **`true`**, and zero values are **`false`**
- Example:

```
if ( payCode == 4 )
    cout << "You get a bonus!" << endl;
```

  - Checks the paycode, and if it is **`4`** then a bonus is awarded
- If **==** was replaced with **=**

```
if ( payCode = 4 )
   cout << "You get a bonus!" << endl;
```

  - Sets **`paycode`** to **`4`**
  - **`4`** is nonzero, so the expression is **`true`** and a bonus is awarded, regardless of **`paycode`**.

6. Confusing Equality (==) and Assignment (=) Operators

- Lvalues
  - Expressions that can appear on the left side of an equation
  - Their values can be changed
  - Variable names are a common example (as in `x = 4;`)
- Rvalues
  - Expressions that can only appear on the right side of an equation
  - Constants, such as numbers (i.e. you cannot write `4 = x;`)
- Lvalues can be used as rvalues, but not vice versa