Chapter 2

Expression & Operators

Outlines

- ★ Expressions
- ★ Operators
- ★ Operator Precedence
- ★ Data Type Conversion

2.1 Expressions

- ➔ An expression is a combination of operators, constants and variables.
- ➔ An expression may consist of one or more operands, and zero or more operators to produce a value.
- ➔ Some expressions are
  - ◆ Arithmetic Expression. (E.g 3 + 4)
  - ◆ Relational Expression. (E.g 3 > 4)
  - ◆ Logical Expression. (E.g 3 > 4 && 4 < 9)

2.2 Operators

- ➔ Operators are symbols that perform operations on variables and values.
- ➔ We will discuss these operators.
  - ◆ Arithmetic Operator
  - ◆ Comparison Operator
  - ◆ Logical Operator
  - ◆ Assignment Operator
  - ◆ Increment/Decrement Operator

2.2.1 Arithmetic Operator

Arithmetic operators are used to perform common mathematical operations.

Operator	Description	example
+	Addition	4 + 3
-	Subtraction	4 - 3
*	Multiplication	4 * 3
/	Division	4 / 3
%	Modulo Operation (Remainder after division)	4 % 3

Example

```
#include <iostream>
using namespace std;

int main() {
   int a, b;
   a = 4;
   b = 2;

   // printing the sum of a and b
   cout << "a + b = " << (a + b) << endl;

   // printing the difference of a and b
   cout << "a - b = " << (a - b) << endl;
```

```
    // printing the product of a and b
    cout << "a * b = " << (a * b) << endl;

    // printing the division of a by b
    cout << "a / b = " << (a / b) << endl;

    // printing the modulo of a by b
    cout << "a % b = " << (a % b) << endl;

    return 0;
}
```

```
Output:

a + b = 6
a - b = 2
a * b = 8
a / b = 2
a % b = 0
```

2.2.2 Relational or Comparison Operator

➔ Comparison operators are used to compare two values.

➔ The return value of a comparison is either true (1) or false (0).

Operator	Description	Example
==	Is equal to	3 == 4
!=	Not equal to	3 != 4
>	Greater than	3 > 4
<	Less than	3 < 4
>=	Greater than or equal	3 >= 4
<=	Less than or equal	3 <= 4

Example

```
#include <iostream>
using namespace std;

int main() {
    int a, b;
    a = 3;
    b = 5;
    bool result;

    result = (a == b);   // false
    cout << "3 == 5 is " << result << endl;

    result = (a != b);  // true
    cout << "3 != 5 is " << result << endl;

    result = a > b;   // false
    cout << "3 > 5 is " << result << endl;
```

```
    result = a < b;   // true
    cout << "3 < 5 is " << result << endl;

    result = a >= b;  // false
    cout << "3 >= 5 is " << result << endl;

    result = a <= b;  // true
    cout << "3 <= 5 is " << result << endl;

    return 0;
}
```

```
Output:

3 == 5 is 0
3 != 5 is 1
3 > 5 is 0
3 < 5 is 1
3 >= 5 is 0
3 <= 5 is 1
```

2.2.3 Logical Operator

- ➔ Logical operators are used to check whether an expression is true or false.
- ➔ If the expression is true, it returns 1 whereas if the expression is false, it returns 0.

Operator	Description	example
&&	Logical AND. True only if all the operands are true.	expression1 && expression2
||	Logical OR. True if at least one of the operands is true.	expression1 || expression2
!	Logical NOT. True only if the operand is false.	!expression

Example

```
#include <iostream>
using namespace std;

int main() {
    bool result;

    result = (3 != 5) && (3 < 5);     // true
    cout << "(3 != 5) && (3 < 5) is " << result << endl;

    result = (3 == 5) && (3 < 5);    // false
    cout << "(3 == 5) && (3 < 5) is " << result << endl;

    result = (3 == 5) && (3 > 5);    // false
    cout << "(3 == 5) && (3 > 5) is " << result << endl;

    result = (3 != 5) || (3 < 5);    // true
    cout << "(3 != 5) || (3 < 5) is " << result << endl;

    result = (3 != 5) || (3 > 5);    // true
```

```
    cout << "(3 != 5) || (3 > 5) is " << result << endl;

    result = (3 == 5) || (3 > 5);    // false
    cout << "(3 == 5) || (3 > 5) is " << result << endl;

    result = !(5 == 2);    // true
    cout << "!(5 == 2) is " << result << endl;

    result = !(5 == 5);    // false
    cout << "!(5 == 5) is " << result << endl;

    return 0;
}
```

```
Output:

(3 != 5) && (3 < 5) is 1
(3 == 5) && (3 < 5) is 0
(3 == 5) && (3 > 5) is 0
(3 != 5) || (3 < 5) is 1
(3 != 5) || (3 > 5) is 1
(3 == 5) || (3 > 5) is 0
!(5 == 2) is 1
!(5 == 5) is 0
```

2.2.4 Assignment Operator

Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	a = 7	a = 7
+=	a += 2	a = a + 2
-=	a -= 2	a = a - 2
*=	a *= 2	a = a * 2
/=	a /= 2	a = a / 2
%=	a %= 2	a = a % 2

Example

```
#include <iostream>
using namespace std;

int main() {
    int a, b;

    // 7 is assigned to a
    a = 7;

    // 2 is assigned to b
    b = 2;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "\nAfter a += b;" << endl;

    // assigning the sum of a and b to a
    a += b;  // a = a +b
    cout << "a = " << a << endl;
```

```
Outcome:

a = 7
b = 2

After a += b;
a = 9
```

```
   return 0;
}
```

2.2.5 Increment/Decrement Operator

- ➔ The increment operator ++ increases the value of a variable by 1.
- ➔ Similarly, the decrement operator -- decreases the value of a variable by 1.
- ➔ **prefix(++var)** - the value of var is incremented by 1; then it returns the value.
- ➔ **postfix(var++)** - the original value of var is returned first; then var is incremented by 1.

Example

```
#include <iostream>

using namespace std;

int main() {
  int var1 = 5, var2 = 5;

  // 5 is displayed
  // Then, var1 is increased to 6.
  cout << var1++ << endl;

  // var2 is increased to 6
  // Then, it is displayed.
  cout << ++var2 << endl;

  return 0;
}
```

```
Output:

5
6
```

2.3 Operator Precedence

- ➔ You should use () when a expression has more than one operator

Order	Operators
first	()
second	*, / , %
thrid	+, -

Example

```
#include <iostream>
using namespace std;

main() {
  int a = 20;
```

```
  int b = 10;
  int c = 15;
  int d = 5;
  int e;

  e = (a + b) * c / d;      // ( 30 * 15 ) / 5
  cout << "Value of (a + b) * c / d is :" << e << endl ;

  e = ((a + b) * c) / d;    // (30 * 15 ) / 5
  cout << "Value of ((a + b) * c) / d is  :" << e << endl ;

  e = (a + b) * (c / d);   // (30) * (15/5)
  cout << "Value of (a + b) * (c / d) is  :" << e << endl ;

  e = a + (b * c) / d;     //  20 + (150/5)
  cout << "Value of a + (b * c) / d is  :" << e << endl ;

  return 0;
}
```

Output:

```
Value of (a + b) * c / d is :90
Value of ((a + b) * c) / d is  :90
Value of (a + b) * (c / d) is  :90
Value of a + (b * c) / d is  :50
```

2.4 Data Type Conversion

- ➔ C++ allows us to convert data of one type to that of another.
- ➔ This is known as type conversion.
- ➔ There are two types of type conversion: Implicit and Explicit

2.4.1 Implicit Type Conversion

- ➔ C++ compilers can convert the value of one data type into another by default.

Example - Convert int to float with Implicit Casting

```
// Working of implicit type-conversion

#include <iostream>
using namespace std;
int main ()
{
// assign the integer value
int num1 = 25;
// declare a float variable
float num2;
// convert int value into float variable using implicit conversion
num2 = num1;
cout <<  " The value of num1 is: " << num1 << endl;
cout <<  " The value of num2 is: " << num2 << endl;
return 0;
```

```
}
```

```
Output:

The value of num1 is: 25
 The value of num2 is: 25
```

2.4.2 Explicit Type Conversion

➔ User has to force the compiler to convert the one data type value to another data type value by using the type casting operator. [**(data type) expression**]

Example - Convert float to int with Explicit Casting

```
#include <iostream>
using namespace std;
int main ()
{
// declare a float variable
float num2;
// initialize an int variable
int num1 = 25;

// convert data type from int to float
num2 = (float) num1;
cout << " The value of int num1 is: " << num1 << endl;
cout << " The value of float num2 is: " << num2 << endl;
return 0;
}
```

```
Output:

The value of int num1 is: 25
The value of float num2 is: 25
```

Thank you