Operators in C++

Dr Ian Cornelius

Hello

Hello (1)

Learning Outcomes

1. Understand the different operators built-in to C++
2. Demonstrate the ability to declare variables and using a variety of operators

Operators

Operators (1)

- You should remember what an operator is from the first year programming module
- To recap:
  - an operator is a character that represents an action of some sort
  - they are used for performing operations on variables and values (otherwise known as operands)
- C++ has a collection of operators built-in:
  - Arithmetic
  - Assignment
  - Comparison
  - Logical
  - Bitwise

Arithmetic

Arithmetic (1)

- These operators are used with numeric values to perform mathematical operations
- Arithmetic and assignment operators in C++ consist of:
  - Addition
  - Subtraction
  - Division
  - Multiplication
  - Modulus

Arithmetic (2)

Addition

- The addition operator makes use of:
  - plus (+) character
  - plus-equals (+=) characters
- When presented with two values or variables, they will add them together

```
int x = 3;
int additionExample1 = 1 + 2;
int additionExample2 = 1 + x;
int additionExample3 = 2;
```

```
additionExample1 -> 3
additionExample2 -> 4
additionExample3 += x -> 5
```

Arithmetic (3)

Subtraction

- The subtraction operator makes use of:
  - minus (`-`) character
  - minus-equals (`-=`) characters
- When presented with two values or variables, they will subtract them from each other

```
int x = 3;
int subtractionExample1 = 2 - 2;
int subtractionExample2 = x - 1;
int subtractionExample3 = 2;
```

```
subtractionExample1 -> 0
subtractionExample2 -> 2
subtractionExample3 -= x -> -1
```

Arithmetic (4)

Division

- The division operator makes use of:
  - forward slash (`/`) character
  - forward-slash-equals (`/=`) characters
- When presented with two values or variables, a division will be performed between the two variables

```cpp
int x = 3;
int divisionExample1 = 9 / 3;
int divisionExample2 = x / 2;
int divisionExample3 = 9;
```

```
divisionExample1 -> 3
divisionExample2 -> 1
divisionExample3 /= x -> 3
```

Arithmetic (5)

Multiplication

- The multiplication operator makes use of:
  - asterisk (`*`) character
  - asterisk-equals (`*=`) characters
- When presented with two values or variables, a multiplication will be performed between the two variables

```
int x = 3;
int multiplicationExample1 = 9 * 3;
int multiplicationExample2 = x * 2;
int multiplicationExample3 = 9;
```

```
multiplicationExample1 -> 27
multiplicationExample2 -> 6
multiplicationExample3 *= x -> 27
```

Arithmetic (6)

Modulus

- The modulus operator makes use of:
  - percentage (`%`) character
  - percentage-equals (`%=`) characters
- When presented with two values or variables, it will return the remainder of a division calculation

```cpp
int x = 8;
int modulusExample1 = 2 % 4;
int modulusExample2 = x % 2;
int modulusExample3 = 3;
```

```
modulusExample1 -> 2
modulusExample2 -> 0
modulusExample3 %= x -> 3
```

Comparison

Comparison (1)

- These operators are used to compare two values together
- Comparison operators in C++ consist of:
  - Same As
  - Not Equal
  - Greater Than
  - Greater Than or Equal To
  - Lower Than
  - Lower Than or Equal To

Comparison (2)

Same As

- The same as operator makes use of:
    - equal-equal (`==`) characters
- This operator is used to check if one variable is the same as another

```
int x = 3;
int y = 3;
int z = 5;
bool sameAsExample1 = (x == y);
bool sameAsExample2 = (x == z);
```

```
sameAsExample1 -> 1
sameAsExample2 -> 0
```

Comparison (3)

Not Equal

- The not equal operator makes use of:
  - exclamation-equal (`!=`) characters
- This operator is used to check if one variable is **not** the same as another

```
int x = 3;
int y = 3;
int z = 5;
bool notEqualExample1 = (x != y);
bool notEqualExample2 = (x != z);
```

```
notEqualExample1 -> 0
notEqualExample2 -> 1
```

Comparison (4)

Greater Than

- The greater than operator makes use of:
  - right-angle-bracket (>) character
- This operator is used to check if one variable is greater than another

```
int x = 3;
int y = 2;
int z = 1;
bool greaterThanExample1 = (x > y);
bool greaterThanExample2 = (z > x);
```

```
greaterThanExample1 -> 1
greaterThanExample2 -> 0
```

Comparison (5)

Greater Than or Equal To

- The greater than or equal to operator makes use of:
  - right-angle-bracket-equal (>=) characters
- This operator is used to check if one variable is greater than another
  - *or* whether it is equal to it

```cpp
int x = 3;
int y = 2;
int z = 3;
bool greaterThanEqualToExample1 = (x >= y);
bool greaterThanEqualToExample2 = (z > x);
bool greaterThanEqualToExample3 = (z >= x);
```

```
greaterThanEqualToExample1 -> 1
greaterThanEqualToExample2 -> 0
greaterThanEqualToExample3 -> 1
```

Comparison (6)

Less Than

- The less than operator makes use of:
  - left-angle-bracket (<) character
- This operator is used to check if one variable is less than another

```
int x = 3;
int y = 2;
int z = 1;
bool lessThanEqualToExample1 = (x < y);
bool lessThanEqualToExample2 = (z < x);
```

```
lessThanEqualToExample1 -> 0
lessThanEqualToExample2 -> 1
```

Comparison (7)

Less Than or Equal To

- The less than or equal to operator makes use of:
  - left-angle-bracket-equal (<=) characters
- This operator is used to check if one variable is less than another
  - *or* whether it is equal to it

```cpp
int x = 3;
int y = 2;
int z = 3;
bool lessThanEqualToExample1 = (x >= y);
bool lessThanEqualToExample2 = (z > x);
bool lessThanEqualToExample3 = (z >= x);
```

```
lessThanEqualToExample1 -> 1
lessThanEqualToExample2 -> 0
lessThanEqualToExample3 -> 1
```

Logical

Logical (1)

- These operators are used to combine comparison operators
- Logical operators in C++ consist of:
  - And
  - Or
  - Not

Logical (2)

And

- The **and** operator checks whether both comparison operators evaluate to `true`
- Uses two ampersand (`&&`) characters for the logical check
    - if both checks evaluate to `true` then `true` is returned
    - if *one* of the checks evaluate to `false` then `false` will be returned

```
bool andExample1 = (6 > 5 && 6 < 10);
bool andExample2 = (6 > 7 && 6 < 10);
```

```
andExample1 -> 1
andExample2 -> 0
```

Logical (3)

Or

- The `or` operator checks whether one of the comparison operators evaluate to `true`
- Uses two pipe (`||`) characters for the logical check
    - if one of these checks evaluate to `true` then `true` is returned

```
bool orExample1 = (6 > 5 || 6 < 10);
```

```
orExample1 -> 1
```

Logical (4)

Not

- The **not** operator will return the reverse of an evaluated condition
- Uses a single exclamation (`!`) character for the logical check
    - if something is returned as `true` it will then be returned as `false` and vice-versa

```
bool notExample1 = (6 > 5);
bool notExample2 = !(6 > 5);
```

```
notExample1 -> 1
notExample2 -> 0
```

Bitwise

Bitwise (1)

- These operators are used to perform operations on individual bits
  - they can only be used on `char` and `int` data types
- Bitwise operators in C++ consist of:
  - Binary AND
  - Binary OR
  - Binary XOR
  - Binary Complement
  - Binary Shift Left
  - Binary Shift Right

Bitwise (2)

Binary AND

- The **Binary AND** operator uses a single ampersand (`&`) character
    - will return `1` if and only if both of the operands are `1`
    - otherwise it will return `0`

```cpp
int binaryAndExample1 = 12;
int binaryAndExample2 = 25;
int binaryAndExample3 = (binaryAndExample1 & binaryAndExample2);
```

```
binaryAndExample1 -> 12         Binary Form: 00001100
binaryAndExample2 -> 25         Binary Form: 00011001
binaryAndExample3 -> 8          Binary Form: 00001000
```

Bitwise (3)

Binary OR

- The **Binary OR** operator uses a single pipe (`|`) character
  - will return `1` if at least one of the operands is `1`
  - otherwise it will return `0`

```
int binaryOrExample1 = 12;
int binaryOrExample2 = 25;
int binaryOrExample3 = (binaryOrExample1 | binaryOrExample2);
```

```
binaryOrExample1 -> 12          Binary Form: 00001100
binaryOrExample2 -> 25          Binary Form: 00011001
binaryOrExample3 -> 29          Binary Form: 00011101
```

Bitwise (4)

Binary XOR

- The **Binary XOR** operator uses a single caret (`^`) character
  - will return `1` if and only if one of the operands is `1`
  - however, if bother operands are `0`, or both are `1`, it will return `0`

```
int binaryXorExample1 = 12;
int binaryXorExample2 = 25;
int binaryXorExample3 = (binaryXorExample1 ^ binaryXorExample2);
```

```
binaryXorExample1 -> 12         Binary Form: 00001100
binaryXorExample2 -> 25         Binary Form: 00011001
binaryXorExample3 -> 21         Binary Form: 00010101
```

Bitwise (5)

Binary Complement

- The **Binary Complement** operator uses a single tilde (`~`) character
    - wll flip the binary digits, i.e. `1` to `0` and `0` to `1`

```cpp
int binaryCompExample1 = 35;
int binaryCompExample2 = ~binaryCompExample1;
```

```
binaryCompExample1 -> 35                Binary Form: 00100011
binaryCompExample2 -> -36               Binary Form: 11011100
```

Bitwise Operator (6)

Binary Shift Left

- The **Binary Shift Left** operator uses a two left-angle-bracket (`<<`) character
  - it will shift all binary digits to the left by a given number of bits
  - bit positions vacated by the left shift operator are filled with a `0`

```
int binaryShiftLeftExample1 = 32;
for (int i = 1; i <= 2; ++i) {
  binaryShiftLeftExample1 << i;
}
```

```
binaryShiftLeftExample1 -> 32          Binary Form: 00100000
Left Shift by 1: 64                    Binary Form: 01000000
Left Shift by 2: 128                   Binary Form: 10000000
```

Bitwise Operator (7)

Binary Shift Right

- The **Binary Shift Right** operator uses a two right-angle-bracket (`>>`) character
  - it will shift all binary digits to the right by a given number of bits
  - bit positions vacated by the right shift operator are filled with a `0`

```
int binaryShiftRightExample1 = 32;
for (int i = 1; i <= 2; ++i) {
  binaryShiftRightExample1 >> i;
}
```

```
binaryShiftRightExample1 -> 32          Binary Form: 00100000
Right Shift by 1: 16                    Binary Form: 00010000
Right Shift by 2: 8                     Binary Form: 00001000
```

Goodbye

Goodbye (1)

Questions and Support

- Questions? Post them on the **Community Page** on Aula
- Additional Support? Visit the Module Support Page
- Contact Details:
  - Dr Ian Cornelius, ab6459@coventry.ac.uk