# Fundamentals of Programming

Lecture 4

**Chamila Karunatilake**

Department of Information and Communication Technology

Faculty of Technology

University of Sri Jayewardenepura

***chamilakarunatilake@sjp.ac.lk***

## Operators in C

## C Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Miscellaneous Operators

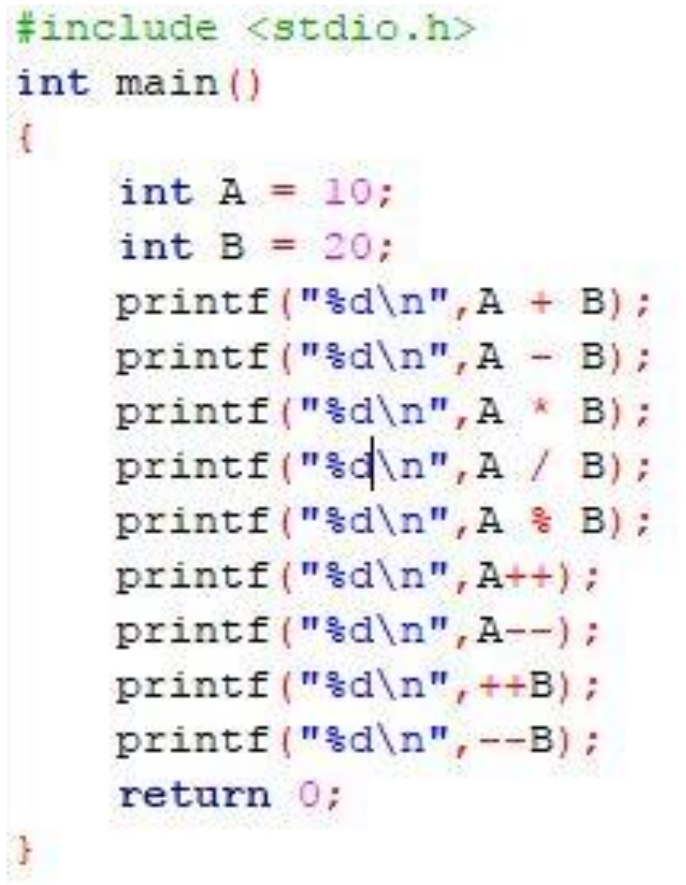
# Arithmetic Operators

C provides arithmetic operators to enable programmers to perform all types of arithmetic.

Arithmetic operator would result an integer or a fraction depending on the operands.

| Operator | Name | Description | Example (A = 10 , B = 20) |
|---|---|---|---|
| + | Addition | Adds two operands. | A+B |
| - | Subtraction | Subtracts second operand from the first. | A-B |
| * | Multiplication | Multiplies both operands. | A*B |
| / | Division | Divides numerator by de-numerator. | A/B |
| % | Modulus (Remainder) | Modulus Operator and remainder of after an integer division. | A%B |
| ++ | Increment | Increment operator increases the integer value by one. | A++ |
| -- | Decrement | Decrement operator decreases the integer value by one. | A-- |

## Arithmetic Operators



```
#include <stdio.h>
int main()
{
    int A = 10;
    int B = 20;
    printf("%d\n",A + B);
    printf("%d\n",A - B);
    printf("%d\n",A * B);
    printf("%d\n",A / B);
    printf("%d\n",A % B);
    printf("%d\n",A++);
    printf("%d\n",A--);
    printf("%d\n",++B);
    printf("%d\n",--B);
    return 0;
}
```

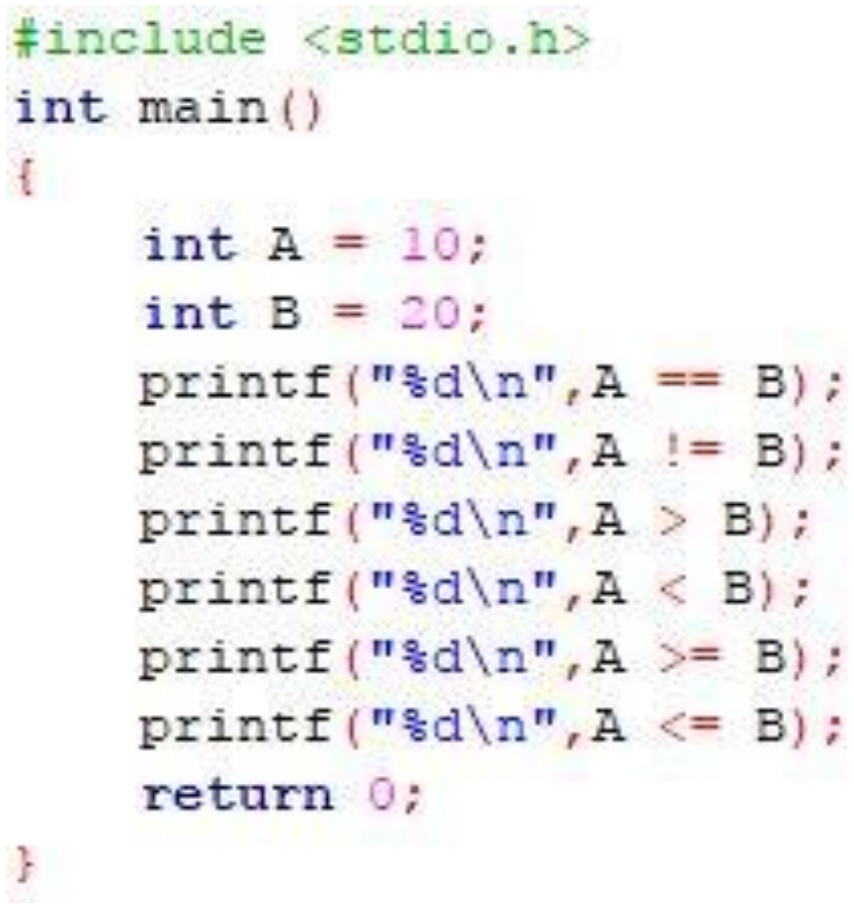
## Relational Operators

Relational operators are utilized when comparing two values(variables or constants).

They would result binary value(true or false) depending on the operands.  Since there is no Boolean values in C, result is either 1 or 0.

| Operator | Name | Description | Example (**A** = 10 , **B** = 20 ) |
|---|---|---|---|
| == | Equals | Checks if the values of two operands are equals. | A == B |
| != | Not Equals | Checks if the values of two operands are not equals. | A != B |
| > | Grater Than | Checks if the value of left operand is greater than the value of right operand. | A > B |
| < | Less Than | Checks if the value of left operand is less than the value of right operand. | A < B |
| >= | Grater Than or Equals | Checks if the value of left operand is greater than or equal to the value of right operand. | A >= B |
| <= | Less Than or Equals | Checks if the value of left operand is less than or equal to the value of right operand. | A <= B |

## Relational Operators



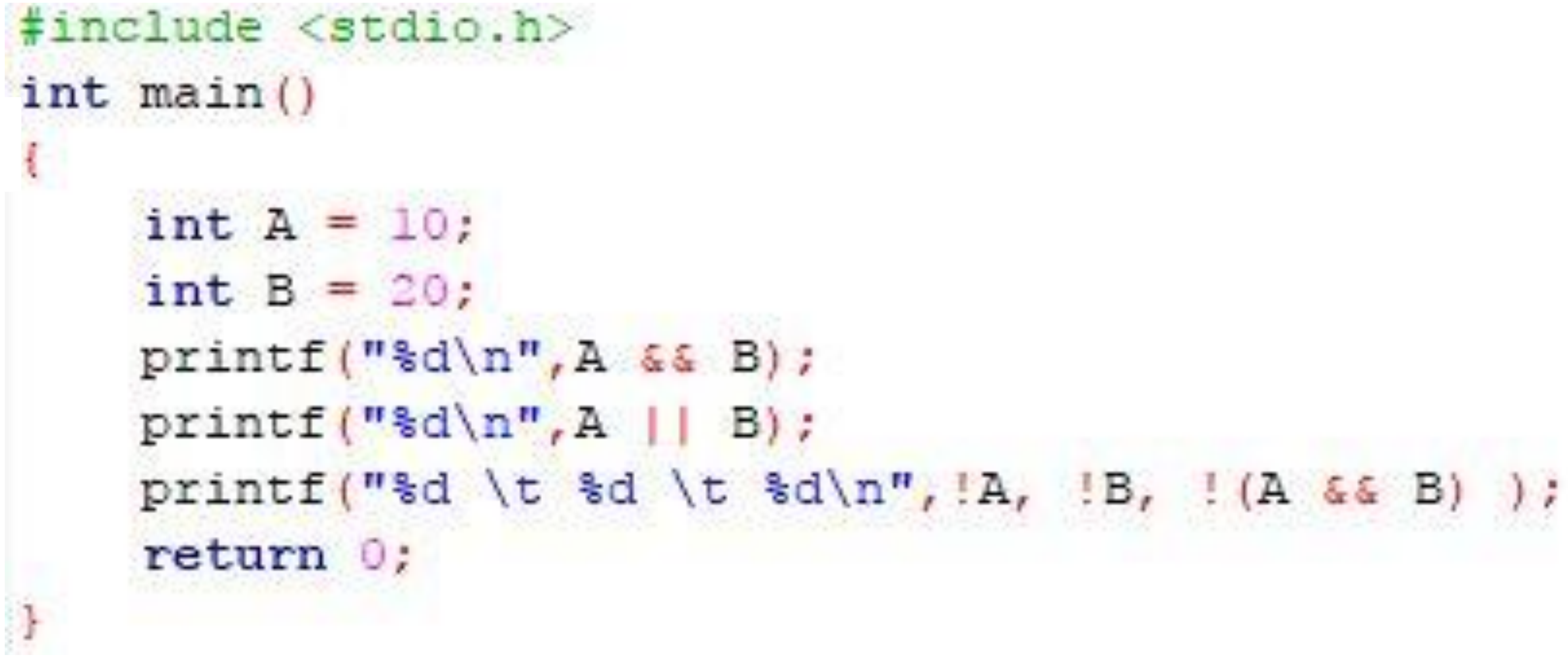
```
#include <stdio.h>
int main()
{
    int A = 10;
    int B = 20;
    printf("%d\n",A == B);
    printf("%d\n",A != B);
    printf("%d\n",A > B);
    printf("%d\n",A < B);
    printf("%d\n",A >= B);
    printf("%d\n",A <= B);
    return 0;
}
```

# Logical Operators

Logical operators always result binary value(true or false) depending on the operands. Since there is no Boolean values in C, result is either 1 or 0.

| Operator | Name | Description | Example<br>(**A** = 10 , **B** = 20) |
|---|---|---|---|
| && | AND | If both the operands are non-zero, then the condition becomes true and results 1 | A && B |
| \|\| | OR | If any of the two operands is non-zero, then the condition becomes true and results 1 | A \|\| B |
| ! | NOT | It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false | !A<br>!B<br>!(A && B) |

## Logical Operators



```
#include <stdio.h>
int main()
{
    int A = 10;
    int B = 20;
    printf("%d\n",A && B);
    printf("%d\n",A || B);
    printf("%d \t %d \t %d\n",!A, !B, !(A && B) );
    return 0;
}
```

# Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation.

The truth table for &, | and ^ can be stated as follows.

| A | B | A&B | A\|B | A^B |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

## Bitwise Operators

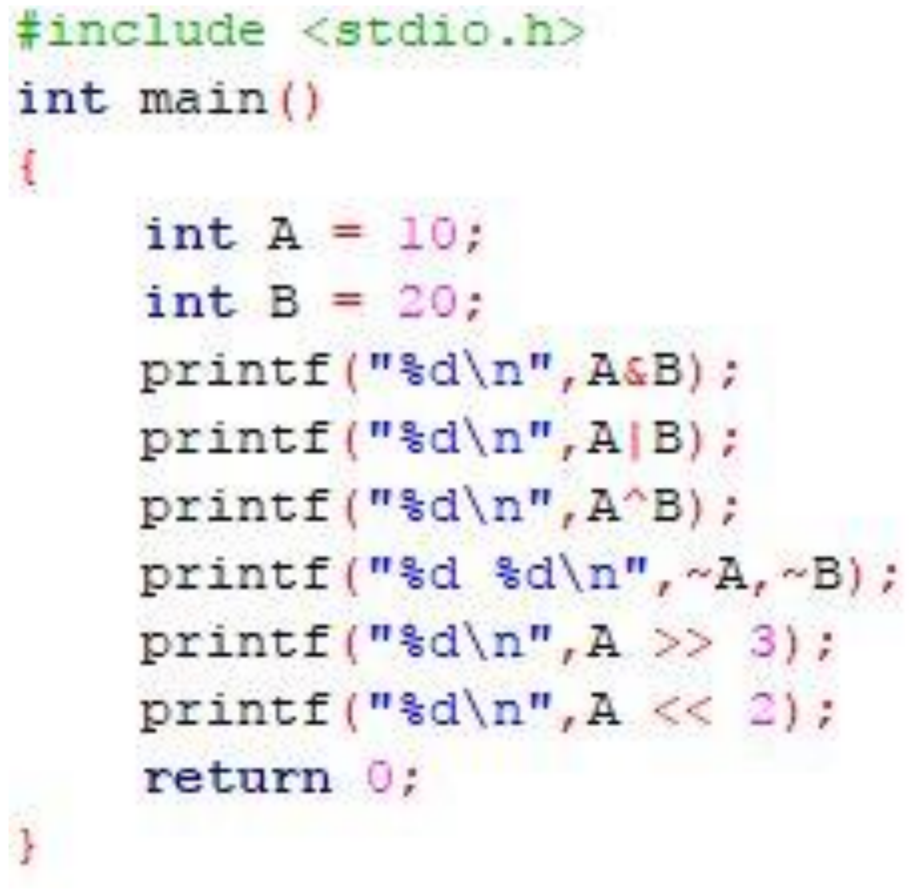
Assume A = 60 and B = 13. In binary format, they will be as follows:

```
A = 0011 1100

B = 0000 1101

-----------------

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A =   1100 0011
```

## Bitwise Operators

| Operator | Name | Description | Example<br>**(A** = 10 , **B** = 20) |
|---|---|---|---|
| & | Bitwise AND | Copies a bit to the result if it exists in both operands | A & B |
| \| | Bitwise OR | Copies a bit if it exists in either operand | A \| B |
| ^ | Bitwise XOR | Copies the bit if it is set in one operand but not both | B ^ 10 |
| ~ | Bitwise Complement | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. The final result will be given as 2s' complement. | ~A<br>~B |
| << | Bitwise Left Shift | The left operand's value is moved left by the number of bits specified by the right operand | A << 2 |
| >> | Bitwise Right shift | The left operands value is moved right by the number of bits specified by the right operand | A >> 2 |

## Bitwise Operators



```
#include <stdio.h>
int main()
{
    int A = 10;
    int B = 20;
    printf("%d\n",A&B);
    printf("%d\n",A|B);
    printf("%d\n",A^B);
    printf("%d %d\n",~A,~B);
    printf("%d\n",A >> 3);
    printf("%d\n",A << 2);
    return 0;
}
```

## Assignment Operators

| Operator | Name | Description | Example<br>(**A** = 10 , **B** = 20) |
|---|---|---|---|
| = | Assignment | Assigns values from right side operands to left side operand | C = A +B |
| += | Add and Assignment | Adds the right operand to the left operand and assign the result to the left operand | A += 10 |
| -= | Subtract and Assignment | Subtracts the right operand from the left operand and assigns the result to the left operand | B -= 10 |
| *= | Multiply and Assignment | Multiplies the right operand with the left operand and assigns the result to the left operand | A *= 2 |
| /= | Divide and Assignment | Divides the left operand with the right operand and assigns the result to the left operand | A /= 2 |
| %= | Modulus and Assignment | Takes modulus using two operands and assigns the result to the left operand | A %= 2 |

# Assignment Operators

| Operator | Name | Description | Example<br>**(A** = 10 , **B** = 20) |
|---|---|---|---|
| <<= | Left shift and Assignment | Performs right shift to left operand using right operand and assigns the result to the left operand | A <<= 2 |
| >>= | Right shift and Assignment | Performs left shift to left operand using right operand and assigns the result to the left operand | B >>= 3 |
| &= | Bitwise AND Assignment | Performs bitwise AND between left and right operands and assigns the result to the left operand | A &= B |
| \|= | Bitwise OR Assignment | Performs bitwise OR between left and right operands and assigns the result to the left operand | A \|= B |
| ^= | Bitwise XOR Assignment | Performs bitwise XOR between left and right operands and assigns the result to the left operand | A ^= B |

## Miscellaneous Operators

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

| **Operator** | **Description** | **Example** **(A** = 10 , **B** = 20) |
|---|---|---|
| Sizeof() | Returns the size of a variable. | Sizeof(A) |
| & | Returns the address of a variable. | &A |
| * | Pointer to a variable. | *A |
| ? : | Conditional Expression | If Condition is true ? then value X : otherwise, value Y |

## Operators Precedence in C

- Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated.
- Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

Example:

```
int x = 7 + 3 * 2;
```

- Here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.
- In next slide, the operator precedence table is given. operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.
- Within an expression, higher precedence operators will be evaluated first.

# Operators Precedence in C

| Category | Operators | 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 |

# Operators Precedence in C

| Category | Operators | Associativity |
|---|---|---|
| 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 |

# Questions?