# Operators & Expressions

## CHAPTER 24

SURESH TECHS

C PROGRAMMING COURSE

```c
#include<stdio.h>
int main(){
    int a;
    int b;
    a = 10;
    b = 30;
    int sum = a+b;
    printf("%d\n",Sum);
    printf("%d, %d\n",a,b);
    a = 90;
    int sub = a-b;
    printf("%d",sub);
}
```

# Operator (+)

# Operands(a,b)

1. Operators are that **symbols** which work on **operands**.
2. Operator is used to perform specific **mathematical or logical** computations on the operands and it **reduces a single value**.

# Operation types

```c
#include<stdio.h>
int main(){
    int a;
    int b;
    a = 10;
    b = 30;
    int sum = a+b;
    printf("%d\n",Sum);
    printf("%d, %d\n",a,b);
    a = 90;
    int sub = a-b;
    printf("%d",sub);
}
```

**Binary operation – binary operator**

**a++**
**++a**
**b--**
**--b**

**Unary operation – unary operator**

**Ternary operation – ternary operator**

# Operators

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Ternary or Conditional Operators
6. Assignment Operators
7. Increment and decrement Operators
8. Special Operators

# Arithmetic Operators

**Used to perform arithmetic operations**

| Operator | Description | Usage |
|---|---|---|
| + | Adds two operands | a+b |
| - | Subtracts the second operand from the first | a-b |
| * | Multiplies both operands | a*b |
| / | Divides numerator by de-numerator | a/b |
| % | return remainder, after an integer division | a%b |

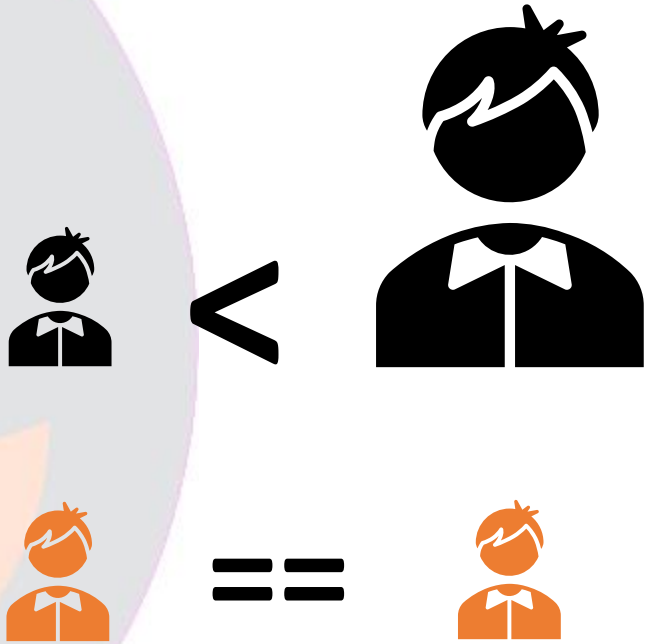
```c
#include<stdio.h>
int main(){
    //Arithmetic operations
    int a = 17;
    int b = 4;
    int sum = a+b;
    int diff = a-b;
    int diff1 = b-a;
    int mul = a*b;
    int div = a/b;
    int modulo = a%b;
    printf("Sum = %d\n",sum);
    printf("Diff = %d\n",diff);
    printf("Diff1 = %d\n",diff1);
    printf("Multiplication = %d\n",mul);
    printf("Div = %d\n",div);
    printf("Modulo = %d\n",modulo);
    return 0;
}
```

```
Sum = 21
Diff = 13
Diff1 = -13
Multiplication = 68
Div = 4
Modulo = 1
```

# Relational operators

**Helps in finding the relationship between the operands**

| Operator | Name | Usage |
|---|---|---|
| == | Equality Operator | checks if a == b |
| != | Not equal to | checks if a != b |
| < | Less than | checks if a < b |
| > | Greater than | checks if a > b |
| <= | Less than or equal to | checks if a<=b |
| >= | Greater than or equal to | checks if a>=b |




**Result is either true or false**

```c
#include<stdio.h>
int main(){
    //Relational operators
    int a = 11;
    int b = 5;
    int c = 6;
    printf("%d\n",a==b);
    printf("%d\n",b==c);
    printf("%d\n",a==b+c);
    printf("%d\n",a>b);
    printf("%d\n",c>b);
    printf("%d\n",a<b);
    printf("%d\n",a<c);
    printf("%d\n",a!=b);
    printf("%d\n",a!=b+c);
    printf("%d\n",a<=b);
    printf("%d\n",b<=b);
    printf("%d\n",c>=b);
    printf("%d\n",a>=b);
    return 0;
}
```

```
0
0
1
1
1
0
0
1
0
0
1
1
1
```

# Logical operators

**Used when we want to check more than one condition**

- **Army selection**
- height should be greater than 167cm **and** weight should be less than 60kg

- **T20 cricket match:** Innings is completed if 20 overs completed **or** 10 players were out

# Logical operators

| Operator | Name | Use |
|---|---|---|
| **&&** | logical AND | Returns true if both side operands value is true otherwise returns false |
| **\|\|** | logical OR | Returns true if one of the operands value is true or both of the operands values is true otherwise returns false |
| **!** | logical Not | Returns true if the condition in consideration is not satisfied otherwise returns false |

# Inputs

- The inputs for the logical operators can be a logical expression or a boolean value.
- A logical expression is any expression that performs a relational operation over two values using the relational operators such as greater than, less than, equal not, etc.

# Output

- The output of all the logical operators is a boolean value.

# Logical AND - &&

- It takes two inputs and combines them.
- The result is **true** only when **both the inputs evaluate to be true**, and **false** if **any of the input is false**
- binary operator

- **Note: If the first input evaluates to be false, then the operator will not even consider checking the next input, because the output is going to be false anyway.**

# Logical AND truth table

| Input 1 | Input 2 | Result |
|---|---|---|
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

# Logical OR ||

- It takes two inputs and combines them.
- The result is **true** when one of the two inputs is true. The output of the logical OR operator is false only when both the input values are false
- binary operator

- **Note: If the first input evaluates to be true, then the operator will not even consider checking the next input, because the output is going to be true anyway.**

# Logical OR Truth table

| Input 1 | Input 2 | Result |
|---|---|---|
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

# Logical NOT !

- Unary operator
- Takes one input and returns the complement(opposite) of it
  - Nijam -> Abaddam
  - Abaddam -> Nijam

| **Input** | **Result** |
|---|---|
| true | false |
| false | true |

```c
#include<stdio.h>
int main(){
    //Logical operators operators
    int a = 10;
    int b = 5;
    int c = 8;
    printf("===AND===\n");
    printf("%d\n",a>b);
    printf("%d\n",a>c);
    printf("%d\n",a>b && a>c);
    printf("===OR===\n");
    printf("%d\n",a>c);
    printf("%d\n",b>c);
    printf("%d\n",a>c || b>c);
    printf("===NOT===\n");
    printf("%d\n",c>b);
    printf("%d\n",!c>b);
    printf("%d\n",a<b);
    printf("%d",!a<b);
    return 0;
}
```

```
===AND===
1
1
1
===OR===
1
0
1
===NOT===
1
0
0
1
```

Will get better understanding while writing programs

# Bitwise operators

- Used to perform operations at the bit level(0 1)
- Bitwise operator first converts the integer into it's binary representation and then performs the operation

# Truth table for Bitwise operations

| A | B | A & B (Bitwise AND) | A \| B (Bitwise OR) | A ^ B (Bitwise XoR) |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 |

| Operator | Name of Operator | What does it do | How it is used |
|---|---|---|---|
| & | bitwise AND | bitwise AND operator do AND of every corresponding bits of both operands and output 1 (true) if both operands have 1 at that position otherwise 0(false). | a & b |
| \| | bitwise OR | bitwise OR operator do OR operation of every corresponding bits of both operands and output 0 (false) if both operands have 0 at that position otherwise 1(true). | a \| b |
| ^ | bitwise exclusive OR | returns 1 if the corresponding bits of two operands are opposite else 0 | a^b |
| **<<** | **shift left** | **shifts the number of bits to the left side** | **a << 1** |
| **>>** | **shift right** | **shifts the number of bits to the right side** | **a >> 1** |

# Shift operators

**Used to perform binary bit operations**

| Operator | Name | Usage | How it is used |
|---|---|---|---|
| **<<** | Left Shift Operator | shifts the number of bits to the left side | a << 1 |
| **>>** | Right Shift Operator | shifts the number of bits to the right side | a >> 2 |

**We don't use it often**

# Ternary or conditional operator

- Will discuss in the decision making and branching topic

# Assignment operators

- Used to assign values to the operands

```c
#include<stdio.h>
int main(){
    int a;
    int b;
    a = 10;
    b = 30;
    int sum = a+b;
    printf("%d\n",Sum);
    printf("%d, %d\n",a,b);
    a = 90;
    int sub = a-b;
    printf("%d",sub);
}
```

| Operator | Name | What does it do | Usage |
|---|---|---|---|
| = | assignment | assign value of variable b to variable a | a = b |
| += | plus assign | a = a+b (adds values of a to b and assign this value to a) | a += b |
| -= | minus assign | a = a-b (subtracts values of b from a and assign this value to a) | a -= b |
| *= | times assign | a = a*b (Multiplies a with b and assign the value to a) | a *= b |
| /= | div assign | a = a/b (divides a by b and assigns the value to a) | a /= b |
| %= | Mod assign | a = a%b (divides a by b and assigns the value of the remainder to a) | a %= b |

| | | | |
|---|---|---|---|
| <<= | Left Shift Assignment Operator means the left operand is left shifted by right operand value and assigned value to left operand | x<<=y | x=x<<y |
| >>= | Right shift Assignment Operator means the left operand is right shifted by right operand value and assigned value to left operand | x>>=y | x=x>>y |
| &= | Bitwise AND Assignment Operator means does AND on every bit of left operand and right operand and assigned value to left operand | x&=y | x=x&y |
| \|= | Bitwise inclusive OR Assignment Operator means does OR on every bit of left operand and right operand and assigned value to left operand | x\|=y | x=x\|y |
| ^= | Bitwise exclusive OR Assignment Operator means does XOR on every bit of left operand and right operand and assigned value to left operand | x^=y | x=x^y |

```c
#include<stdio.h>
int main(){
    //Assignment operators
    int a = 10;
    //Add 5 to a
    a = a+5; //can also be written as a+=5
    printf("%d\n",a);
    a = a-5; //can also be written as a-=5
    printf("%d\n",a);
    a = a*5; //can also be written as a*=5
    printf("%d\n",a);
    a = a/5; //can also be written as a/=5
    printf("%d\n",a);
    a = a%5; //can also be written as a%=5
    printf("%d\n",a);
    return 0;
}
```

```
15
10
50
10
0
```

# Increment/Decrement Operators

- Unary operators
- One operand 😍
- Increment ++
  - Increases the value of the variable by 1
- Decrement --
  - Decreases the value of the variable by 1

# Incrementing a value

```c
#include<stdio.h>
int main(){
    int a = 10;
    a = a+1;
    printf("%d",a);
    return 0;
}
```

```c
#include<stdio.h>
int main(){
    int a = 10;
    //a = a+1;
    a++;
    printf("%d",a);
    return 0;
}
```

# Types of increment operators

- **Prefix increment:** Value of the variable first increases by 1 and then the variable is used inside the expression
- ++a
- **Postfix increment:** The variable is used inside the expression with its original value and then its value is increased by 1
- a++

```c
#include<stdio.h>
int main(){
    int a = 5, b = 6;
    printf("%d\n", ++a);
    printf("%d\n", b++);
    printf("%d\n",b);
    return 0;
}
```

```
6
6
7
```

# Types of decrement operators

- **Prefix decrement:** Value of the variable first decreases by 1 and then the variable is used inside the expression
- --a
- **Postfix decrement:** The variable is used inside the expression with its original value and then its value is decreased by 1
- a--

```c
#include<stdio.h>
int main(){
    int a = 5, b = 6;
    printf("%d\n", ++a);
    printf("%d\n", b++);
    printf("%d\n",b);
    printf("%d\n", --a);
    printf("%d\n", b--);
    printf("%d\n",b);
    return 0;
}
```

```
6
6
7
5
7
6
```

# Special Operators

| Operator | Name of Operator | What it does | How it is used |
|---|---|---|---|
| **,** | Comma | Used to link the related expressions together | value = (x=10, y=5) |
| **&** | Address Operator | Returns the address of the variable. | &a |
| ***** | Pointer Operator | Pointer to a variable | *a |
| **.** | Member Operator | To select data members and functions | a.value |
| **->** | Member Operator | To select data members and functions | a->value |
| **sizeof()** | sizeof | Returns the size of a variable | mSize = sizeof(a) |

# Expressions

- An expression in C is a **combination of operands and operators** – it computes to a single value

```c
#include<stdio.h>
int main(){
    int a = 10;
    int b = 20;
    int result = a+2*100/b;
    printf("%d",result);
    return 0;
}
```

```
20
```

# Types of expressions

1. Arithmetic Expressions
2. Relational Expressions
3. Logical Expressions
4. Bitwise Expressions

# What next?

- Type casting