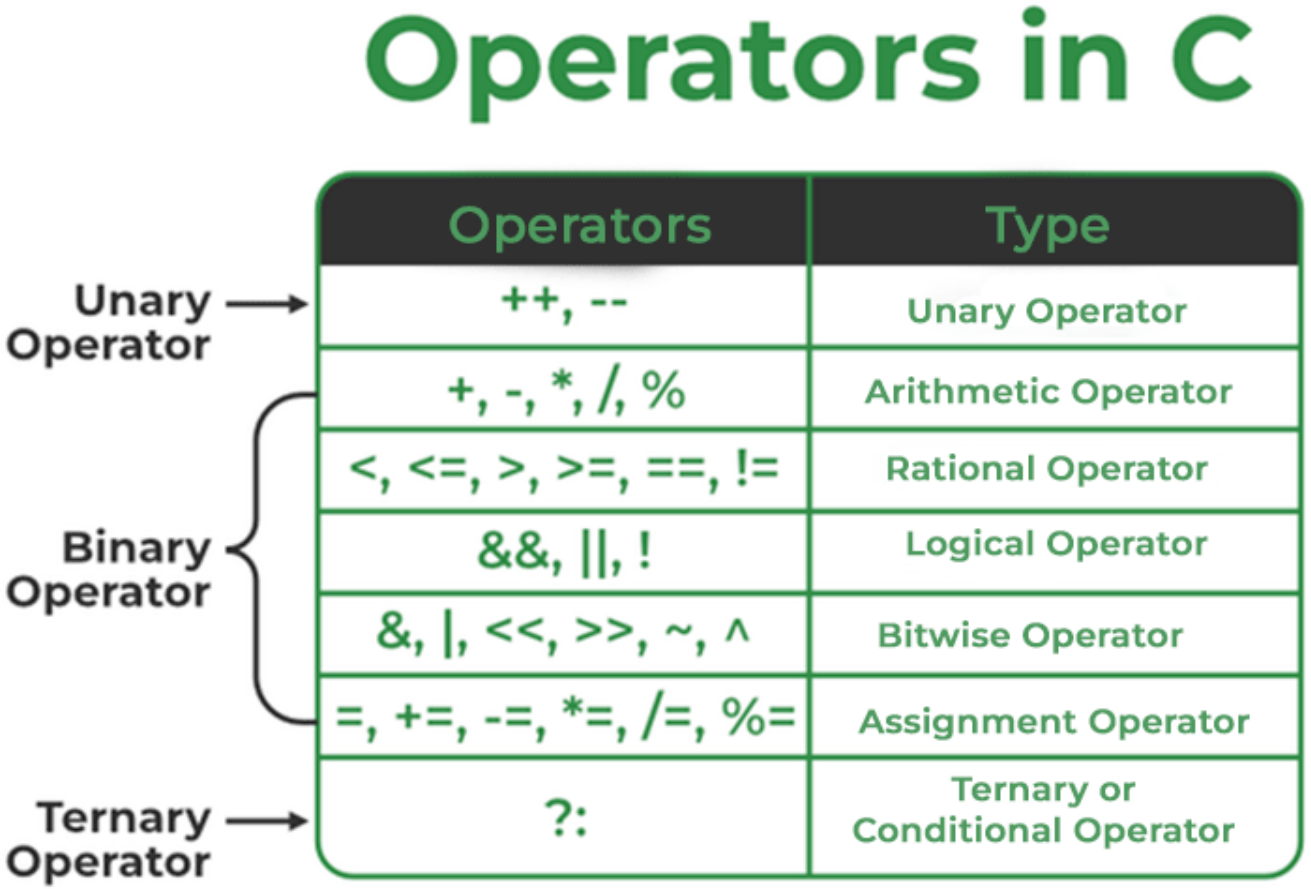
C - Operators

Pratik Shah
II – Sem CSE (DIV A)
02/01/25 to 21/04/25

What is a C Operator?

An operator in C can be defined as the symbol that helps us to perform some specific **mathematical, relational, bitwise, conditional, or logical computations on values and variables. The values and variables used with operators are called operands.** So we can say that the operators are the symbols that perform operations on operands.



Types of Operators in C

C language provides a wide range of operators that can be classified into **6 types** based on their functionality:

1. **Arithmetic Operators**
2. **Relational Operators**
3. **Logical Operators**
4. **Bitwise Operators**
5. **Assignment Operators**
6. **Other Operators**

1. Arithmetic Operations in C

The arithmetic operators are used to perform **arithmetic/mathematical operations** on operands.
There are 9 arithmetic operators in C language

S. No.	Symbol	Operator	Description	Syntax
1	+	Plus	Adds two numeric values.	a + b
2	–	Minus	Subtracts right operand from left operand.	a – b
3	*	Multiply	Multiply two numeric values.	a * b
4	/	Divide	Divide two numeric values.	a / b
5	%	Modulus	Returns the remainder after diving the left operand with the right operand.	a % b
6	+	Unary Plus	Used to specify the positive values.	+a
7	–	Unary Minus	Flips the sign of the value.	-a
8	++	Increment	Increases the value of the operand by 1.	a++
9	—	Decrement	Decreases the value of the operand by 1.	a–

Example of C Arithmetic Operators

```
// C program to illustrate the arithmetic operators
#include <stdio.h>
void main()
{

    int a = 25, b = 5;
    printf("a + b = %d\n", a + b);
    printf("a - b = %d\n", a - b);
    printf("a * b = %d\n", a * b);
    printf("a / b = %d\n", a / b);
    printf("a % b = %d\n", a % b);
    printf("+a = %d\n", +a);
    printf("-a = %d\n", -a);
    printf("a++ = %d\n", a++);
    printf("a-- = %d\n", a--);
}
```

2. Relational Operators in C

The relational operators in C are used for the comparison of the two operands. All these operators are binary operators that **return true or false** values as the result of comparison.
These are a total of **6 relational operators** in C

S. No.	Symbol	Operator	Description	Syntax
1	**<**	**Less than**	Returns true if the left operand is less than the right operand. Else false	**a < b**
2	**>**	**Greater than**	Returns true if the left operand is greater than the right operand. Else false	**a > b**
3	**<=**	**Less than or equal to**	Returns true if the left operand is less than or equal to the right operand. Else false	**a <= b**
4	**>=**	**Greater than or equal to**	Returns true if the left operand is greater than or equal to right operand. Else false	**a >= b**
5	**==**	**Equal to**	Returns true if both the operands are equal.	**a == b**
6	**!=**	**Not equal to**	Returns true if both the operands are NOT equal.	**a != b**

Example of C Relational Operators

```
// C program to illustrate the relational operators
#include <stdio.h>
void main()
{

    int a = 25, b = 5;
    printf("a < b  : %d\n", a < b);
    printf("a > b  : %d\n", a > b);
    printf("a <= b: %d\n", a <= b);
    printf("a >= b: %d\n", a >= b);
    printf("a == b: %d\n", a == b);
    printf("a != b : %d\n", a != b);


}
```

3. Logical Operator in C

Logical Operators are used to **combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration.** The result of the operation of a logical operator is a Boolean value either **true** or **false**.

S. No.	Symbol	Operator	Description	Syntax
1	**&&**	**Logical AND**	Returns true if both the operands are true.	**a && b**
2	**||**	**Logical OR**	Returns true if both or any of the operand is true.	**a || b**
3	**!**	**Logical NOT**	Returns true if the operand is false.	**!a**

Example of C Logical Operators

```
// C program to illustrate the logical operators
#include <stdio.h>
void main()
{

    int a = 25, b = 5;
    printf("a && b : %d\n", a>20 && b>2);
    printf("a || b : %d\n", a>20 || b>10);
    printf("!a: %d\n", !(a>15));
}
```

4. Bitwise Operators in C

The Bitwise operators are used to perform **bit-level operations on the operands**. **The operators are first converted to bit-level and then the calculation is performed on the operands. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing.**

There are **6 bitwise operators** in C:

S. No.	Symbol	Operator	Description	Syntax
1	&	Bitwise AND	Performs bit-by-bit AND operation and returns the result.	a & b
2	|	Bitwise OR	Performs bit-by-bit OR operation and returns the result.	a | b
3	^	Bitwise XOR	Performs bit-by-bit XOR operation and returns the result.	a ^ b
4	~	Bitwise First Complement	Flips all the set and unset bits on the number.	~a
5	<<	Bitwise Leftshift	Shifts the number in binary form by one place in the operation and returns the result.	a << b
6	>>	Bitwise Rightshilft	Shifts the number in binary form by one place in the operation and returns the result.	a >> b

Example of C Bitwise Operators

```
// C program to illustrate the Bitwise operators
#include <stdio.h>
void main()
{

    int a = 25, b = 5;
    printf("a & b: %d\n", a & b);
    printf("a | b: %d\n", a | b);
    printf("a ^ b: %d\n", a ^ b);
    printf("~a: %d\n", ~a);
    printf("a >> b: %d\n", a >> b);
    printf("a << b: %d\n", a << b);
}
```

5. Assignment Operators in C

Assignment operators are used to assign value to a variable.
In C, there are **11 assignment operators** :

S. No.	Symbol	Operator	Description	Syntax
1	=	Simple Assignment	Assign the value of the right operand to the left operand.	a = b
2	+=	Plus and assign	Add the right operand and left operand and assign this value to the left operand.	a += b
3	-=	Minus and assign	Subtract the right operand and left operand and assign this value to the left operand.	a -= b
4	*=	Multiply and assign	Multiply the right operand and left operand and assign this value to the left operand.	a *= b
5	/=	Divide and assign	Divide the left operand with the right operand and assign this value to the left operand.	a /= b
6	%=	Modulus and assign	Assign the remainder in the division of left operand with the right operand to the left operand.	a %= b
7	&=	AND and assign	Performs bitwise AND and assigns this value to the left operand.	a &= b
8	|=	OR and assign	Performs bitwise OR and assigns this value to the left operand.	a |= b
9	^=	XOR and assign	Performs bitwise XOR and assigns this value to the left operand.	a ^= b
10	>>=	Rightshift and assign	Performs bitwise Rightshift and assign this value to the left operand.	a >>= b
11	<<=	Leftshift and assign	Performs bitwise Leftshift and assign this value to the left operand.	a <<= b

Example of C Assignment Operators

```
// C program to illustrate the Assignment operators
#include <stdio.h>
void main()
{
    int a = 25, b = 5;
    printf("a = b: %d\n", a = b);
    printf("a += b: %d\n", a += b);
    printf("a -= b: %d\n", a -= b);
    printf("a *= b: %d\n", a *= b);
    printf("a /= b: %d\n", a /= b);
    printf("a %%= b: %d\n", a %= b);
    printf("a &= b: %d\n", a &= b);
    printf("a |= b: %d\n", a |= b);
    printf("a >>= b: %d\n", a >>= b);
    printf("a <<= b: %d\n", a <<= b);
}
```

6. Other Operators

Apart from the above operators, there are some other operators available in C used to perform some specific tasks.

sizeof Operator

It is a compile-time unary operator which can be used to compute the size of its operand.
The result of sizeof is of the unsigned integral type which is usually denoted by size_t.
Basically, the sizeof the operator is used to compute the size of the variable or datatype.

Syntax

sizeof (operand)

Cast Operator

- Casting operators convert one data type to another. **For example, int(2.2000) would return 2.**
- A cast is a special operator that forces one data type to be converted into another.
- **[(type) expression]**.

Syntax

(new_type) operand;

Conditional Operator (? :)

The conditional operator is the only ternary operator in C++.

Here, Expression1 is the condition to be evaluated. If the condition(Expression1) is True then we will execute and return the result of Expression2 otherwise if the condition(Expression1) is false then we will execute and return the result of Expression3.

We may replace the use of if..else statements with conditional operators.

Syntax

operand1 ? operand2 : operand3;

addressof (&) and Dereference (*) Operators

Pointer operator & returns the address of a variable. For example &a; will give the actual address of the variable.

The **pointer operator * is a pointer to a variable.**

For example ***var; will pointer to a variable var.**

Example of Other C Operators

```
// C Program to demonstrate the use of Misc operators
#include <stdio.h>
void main()
{
    int num = 10;
    int* add_of_num = &num;

   printf("sizeof(num) = %d bytes\n", sizeof(num));
    printf("&num = %p\n", &num);
    printf("*add_of_num = %d\n", *add_of_num);
    printf("(10 < 5) ? 10 : 20 = %d\n", (10 < 5) ? 10 : 20);
    printf("(float)num = %f\n", (float)num);
}
```

Operator Precedence and Associativity in C

In C, it is very common for an expression or statement to have multiple operators and in these expression, there should be a fixed order or priority of operator evaluation to avoid ambiguity.

Operator Precedence and Associativity *is the concept that decides which operator will be evaluated first in the case when there are multiple operators present in an expression.*

Precedence	Operator	Description	Associativity
1	()	Parentheses (function call)	left-to-right
	[]	Brackets (array subscript)	left-to-right
	.	Member selection via object name	left-to-right
	->	Member selection via a pointer	left-to-right
	a++ , a–	Postfix increment/decrement (a is a variable)	left-to-right
2	++a , –a	Prefix increment/decrement (a is a variable)	right-to-left
	+ , –	Unary plus/minus	right-to-left
	! , ~	Logical negation/bitwise complement	right-to-left
	(type)	Cast (convert value to temporary value of type)	right-to-left
	*	Dereference	right-to-left
	&	Address (of operand)	right-to-left
	sizeof	Determine size in bytes on this implementation	right-to-left
3	* , / , %	Multiplication/division/modulus	left-to-right
4	+ , –	Addition/subtraction	left-to-right
5	<< , >>	Bitwise shift left, Bitwise shift right	left-to-right
6	< , <=	Relational less than/less than or equal to	left-to-right
	> , >=	Relational greater than/greater than or equal to	left-to-right
7	== , !=	Relational is equal to/is not equal to	left-to-right
8	&	Bitwise AND	left-to-right

9	**^**	Bitwise XOR	left-to-right
10	**|**	Bitwise OR	left-to-right
11	**&&**	Logical AND	left-to-right
12	**||**	Logical OR	left-to-right
13	**?:**	Ternary conditional	right-to-left
14	**=**	Assignment	right-to-left
	+= , -=	Addition/subtraction assignment	right-to-left
	***= , /=**	Multiplication/division assignment	right-to-left
	%= , &=	Modulus/bitwise AND assignment	right-to-left
	^= , |=	Bitwise exclusive/inclusive OR assignment	right-to-left
	<<=, >>=	Bitwise shift left/right assignment	right-to-left
15	**,**	expression separator	left-to-right

References

- https://www.geeksforgeeks.org/operators-in-c/

Thank You !!!!