SNS COLLEGE OF TECHNOLOGY

Coimbatore-35.
An Autonomous Institution

COURSE NAME : 23ITT101 PROGRAMMING IN C AND DATA STRUCTURES

I YEAR/ II SEMESTER

UNIT-I INTRODUCTION TO C

Topic: C Operators and Expressions

Dr.B.Vinodhini
Associate Professor
Department of Computer Science and Engineering

C Operators

Operators in C Language

C language supports a rich set of built-in operators. An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation. Operators are used in programs to manipulate data and variables.

Types of Operators in C

C programming language offers various types of operators having different functioning capabilities.

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment and Decrement Operators
6. Conditional Operator
7. Bitwise Operators
8. Special Operators

C Operators

Operators in C

	Operator	Type
Unary operator →	+ +, - -	Unary operator
Binary operator	+, -, *, /, %	Arithmetic operator
	<, <=, >, >=, ==, !=	Relational operator
	&&, ||, !	Logical operator
	&, |, <<, >>, ~, ^	Bitwise operator
	=, +=, -=, *=, /=, %=	Assignment operator
Ternary operator →	?:	Ternary or conditional operator

C Operators

Types of Operators	Description	Types of Operators	Description
Arithmetic_operators	These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus	Bit wise operators	These operators are used to perform bit operations on given two variables.
Assignment_operators	These are used to assign the values for the variables in C programs.	Conditional (ternary) operators	Conditional operators return one value if condition is true and returns another value is condition is false.
Relational operators	These operators are used to compare the value of two variables.	Increment/decrement operators	These operators are used to either increase or decrease the value of the variable by one.
Logical operators	These operators are used to perform logical operations on the given two variables.	Special operators	&, *, sizeof( ) and ternary operators.

C Operators

Arithmetic Operators (+, -, *, /, %)

The arithmetic operators are the symbols that are used to perform basic mathematical operations like addition, subtraction, multiplication, division and percentage modulo. The following table provides information about arithmetic operators.

Operator	Meaning	Example
+	Addition	10 + 5 = 15
-	Subtraction	10 - 5 = 5
*	Multiplication	10 * 5 = 50
/	Division	10 / 5 = 2
%	Remainder of the Division	5 % 2 = 1

⇒ The addition operator can be used with numerical data types and character data type. When it is used with numerical values, it performs mathematical addition and when it is used with character data type values, it performs concatination (appending).

⇒ The remainder of the division operator is used with integer data type only.

Operators and Expressions/Dr.B.Vinodhini/ASP/CSE/SNSCT

C Operators

Example:

```c
// Working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

C Operators

Output

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

The operators `+`, `-` and `*` computes addition, subtraction, and multiplication respectively as you might have expected.

In normal calculation, `9/4 = 2.25`. However, the output is `2` in the program.

It is because both the variables `a` and b are integers. Hence, the output is also an integer. The compiler neglects the term after the decimal point and shows answer `2` instead of `2.25`.

The modulo operator `%` computes the remainder. When `a=9` is divided by `b=4`, the remainder is `1`. The `%` operator can only be used with integers.

Suppose `a = 5.0`, `b = 2.0`, `c = 5` and `d = 2`. Then in C programming,

```
// Either one of the operands is a floating-point number
a/b = 2.5
a/d = 2.5
c/b = 2.5

// Both operands are integers
c/d = 2
```

C Operators

Relational Operators (<, >, <=, >=, ==, !=)

The relational operators are the symbols that are used to compare two values. That means the relational operators are used to check the relationship between two values. Every relational operator has two results TRUE or FALSE. In simple words, the relational operators are used to define conditions in a program. The following table provides information about relational operators.

Operator	Meaning	Example
<	Returns TRUE if the first value is smaller than second value otherwise returns FALSE	10 < 5 is FALSE
>	Returns TRUE if the first value is larger than second value otherwise returns FALSE	10 > 5 is TRUE
<=	Returns TRUE if the first value is smaller than or equal to second value otherwise returns FALSE	10 <= 5 is FALSE
>=	Returns TRUE if the first value is larger than or equal to second value otherwise returns FALSE	10 >= 5 is TRUE
==	Returns TRUE if both values are equal otherwise returns FALSE	10 == 5 is FALSE
!=	Returns TRUE if both values are not equal otherwise returns FALSE	10 != 5 is TRUE

C Operators

Example:

```c
// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

Operators and Expressions/Dr.B.Vinodhini/ASP/CSE/SNSCT

C Operators

Output:

```
5 == 5 is 1
5 == 10 is 0
5 > 5 is 0
5 > 10 is 0
5 < 5 is 0
5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
5 >= 5 is 1
5 >= 10 is 0
5 <= 5 is 1
5 <= 10 is 1
```

C Operators

Logical Operators (&&, ||, !)

The logical operators are the symbols that are used to combine multiple conditions into one condition. The following table provides information about logical operators.

Operator	Meaning	Example
&&	**Logical AND** - Returns TRUE if all conditions are TRUE otherwise returns FALSE	10 < 5 && 12 > 10 is FALSE
||	**Logical OR** - Returns FALSE if all conditions are FALSE otherwise returns TRUE	10 < 5 || 12 > 10 is TRUE
!	**Logical NOT** - Returns TRUE if condition is FLASE and returns FALSE if it is TRUE	!(10 < 5 && 12 > 10) is TRUE

⇒ **Logical AND** - Returns TRUE only if all conditions are TRUE, if any of the conditions is FALSE then complete condition becomes FALSE.

⇒ **Logical OR** - Returns FALSE only if all conditions are FALSE, if any of the conditions is TRUE then complete condition becomes TRUE.

C Operators

Example:

```c
// Working of logical operators

#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) is %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);

    result = !(a != b);
    printf("!(a != b) is %d \n", result);

    result = !(a == b);
    printf("!(a == b) is %d \n", result);

    return 0;
}
```

C Operators

Output

```
(a == b) && (c > b) is 1
(a == b) && (c < b) is 0
(a == b) || (c < b) is 1
(a != b) || (c < b) is 0
!(a != b) is 1
!(a == b) is 0
```

Explanation of logical operator program

- `(a == b) && (c > 5)` evaluates to 1 because both operands `(a == b)` and `(c > b)` is 1 (true).

- `(a == b) && (c < b)` evaluates to 0 because operand `(c < b)` is 0 (false).

- `(a == b) || (c < b)` evaluates to 1 because `(a = b)` is 1 (true).

- `(a != b) || (c < b)` evaluates to 0 because both operand `(a != b)` and `(c < b)` are 0 (false).

- `!(a != b)` evaluates to 1 because operand `(a != b)` is 0 (false). Hence, !(a != b) is 1 (true).

- `!(a == b)` evaluates to 0 because `(a == b)` is 1 (true). Hence, `!(a == b)` is 0 (false).

C Operators

Increment & Decrement Operators (++ & --)

The increment and decrement operators are called unary operators because both need only one operand. The increment operators adds one to the existing value of the operand and the decrement operator subtracts one from the existing value of the operand. The following table provides information about increment and decrement operators.

Operator	Meaning	Example
++	**Increment** - Adds one to existing value	int a = 5; a++; ⇒ a = 6
--	**Decrement** - Subtracts one from existing value	int a = 5; a--; ⇒ a = 4

The increment and decrement operators are used infront of the operand (++a) or after the operand (a++). If it is used infront of the operand, we call it as **pre-increment** or **pre-decrement** and if it is used after the operand, we call it as **post-increment** or **post-decrement**.

C Operators

Pre-Increment or Pre-Decrement

In the case of pre-increment, the value of the variable is increased by one before the expression evaluation. In the case of pre-decrement, the value of the variable is decreased by one before the expression evaluation. That means, when we use pre-increment or pre-decrement, first the value of the variable is incremented or decremented by one, then the modified value is used in the expression evaluation.

Example Program

```c
#include<stdio.h>
#include<conio.h>

void main(){
    int i = 5,j;

    j = ++i; // Pre-Increment

    printf("i = %d, j = %d",i,j);

}
```

C Operators

Output:

```
 "C:\Users\User\Desktop\New folder\IncrementDecrement\bin\Debug\IncrementDecrement.exe"

i = 6, j = 6

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

C Operators

Post-Increment or Post-Decrement

In the case of post-increment, the value of the variable is increased by one after the expression evaluation. In the case of post-decrement, the value of the variable is decreased by one after the expression evaluation. That means, when we use post-increment or post-decrement, first the expression is evaluated with existing value, then the value of the variable is incremented or decremented by one.

Example Program

```
#include<stdio.h>
#include<conio.h>

void main(){
    int i = 5,j;

    j = i++;  // Post-Increment

    printf("i = %d, j = %d",i,j);

}
```

C Operators

Output:


```
"C:\Users\User\Desktop\New folder\IncrementDecrement\bin\Debug\IncrementDecrement.exe"

i = 6, j = 5


Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.

```

C Operators

Assignment Operators (=, +=, -=, *=, /=, %=)

The assignment operators are used to assign right-hand side value (Rvalue) to the left-hand side variable (Lvalue). The assignment operator is used in different variants along with arithmetic operators. The following table describes all the assignment operators in the C programming language.

Operator	Meaning	Example
=	Assign the right-hand side value to left-hand side variable	A = 15
+=	Add both left and right-hand side values and store the result into left-hand side variable	A += 10 ⇒ A = A+10
-=	Subtract right-hand side value from left-hand side variable value and store the result into left-hand side variable	A -= B ⇒ A = A-B
*=	Multiply right-hand side value with left-hand side variable value and store the result into left-hand side variable	A *= B ⇒ A = A*B
/=	Divide left-hand side variable value with right-hand side variable value and store the result into the left-hand side variable	A /= B ⇒ A = A/B
%=	Divide left-hand side variable value with right-hand side variable value and store the remainder into the left-hand side variable	A %= B ⇒ A = A%B

C Operators

Example:

```c
// Working of assignment operators
#include <stdio.h>
int main()
{
    int a = 5, c;

    c = a;        // c is 5
    printf("c = %d\n", c);
    c += a;       // c is 10
    printf("c = %d\n", c);
    c -= a;       // c is 5
    printf("c = %d\n", c);
    c *= a;       // c is 25
    printf("c = %d\n", c);
    c /= a;       // c is 5
    printf("c = %d\n", c);
    c %= a;       // c = 0
    printf("c = %d\n", c);

    return 0;
}
```

C Operators

Output:

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

C Operators

Bitwise Operators (&, |, ^, ~, >>, <<)

The bitwise operators are used to perform bit-level operations in the c programming language. When we use the bitwise operators, the operations are performed based on the binary values. The following table describes all the bitwise operators in the C programming language.

Let us consider two variables A and B as A = 25 (11001) and B = 20 (10100).

Operator	Meaning	Example
&	the result of Bitwise AND is 1 if all the bits are 1 otherwise it is 0	A & B ⇒ 16 (10000)
|	the result of Bitwise OR is 0 if all the bits are 0 otherwise it is 1	A | B ⇒ 29 (11101)
^	the result of Bitwise XOR is 0 if all the bits are same otherwise it is 1	A ^ B ⇒ 13 (01101)
~	the result of Bitwise once complement is negation of the bit (Flipping)	~A ⇒ 6 (00110)
<<	the Bitwise left shift operator shifts all the bits to the left by the specified number of positions	A << 2 ⇒ 100 (1100100)
>>	the Bitwise right shift operator shifts all the bits to the right by the specified number of positions	A >> 2 ⇒ 6 (00110)

C Operators

Bitwise AND operator &

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Let us suppose the bitwise AND operation of two integers 12 and 25.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
  00001100
& 00011001
  _____
  00001000  = 8 (In decimal)
```

C Operators

Example #1: Bitwise AND

```c
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a&b);
    return 0;
}
```

Output

```
Output = 8
```

C Operators

Conditional Operator (?:)

The conditional operator is also called a **ternary operator** because it requires three operands. This operator is used for decision making. In this operator, first we verify a condition, then we perform one operation out of the two operations based on the condition result. If the condition is TRUE the first option is performed, if the condition is FALSE the second option is performed. The conditional operator is used with the following syntax.

Condition ? TRUE Part : FALSE Part;

Example

A = (10<15)?100:200; ⇒ A value is 100

C Operators

Special Operators (sizeof, pointer, comma, dot, etc.)

The following are the special operators in c programming language.

sizeof operator

This operator is used to find the size of the memory (in bytes) allocated for a variable. This operator is used with the following syntax.

sizeof(variableName);

Example

sizeof(A); ⇒ the result is 2 if A is an integer

Pointer operator (*)

This operator is used to define pointer variables in c programming language.

Comma operator (,)

This operator is used to separate variables while they are declaring, separate the expressions in function calls, etc.

Dot operator (.)

This operator is used to access members of structure or union.

C Expressions

What is an expression?

In any programming language, if we want to perform any calculation or to frame any condition etc., we use a set of symbols to perform the task. These set of symbols makes an expression.

In the C programming language, an expression is defined as follows.

> An expression is a collection of operators and operands that represents a specific value.

In the above definition, an **operator** is a symbol that performs tasks like arithmetic operations, logical operations, and conditional operations, etc.

Operands are the values on which the operators perform the task. Here operand can be a direct value or variable or address of memory location.

C Expressions

Expression Types in C

In the C programming language, expressions are divided into THREE types. They are as follows...

1. Infix Expression

2. Postfix Expression

3. Prefix Expression

The above classification is based on the operator position in the expression.
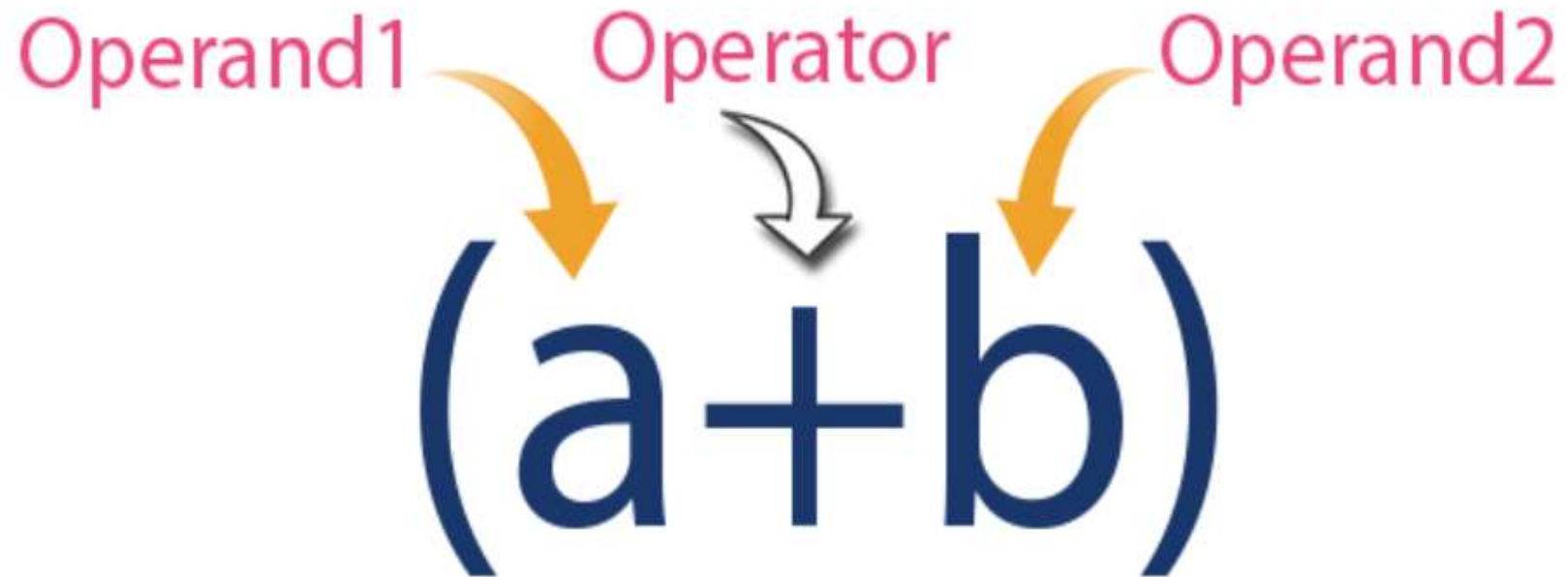
Infix Expression

The expression in which the operator is used between operands is called infix expression.

The infix expression has the following general structure.

> **Operand1 Operator Operand2**

Example

$$Operand1 \quad Operator \quad Operand2$$

$$(a+b)$$

Postfix Expression

The expression in which the operator is used after operands is called postfix expression.

The postfix expression has the following general structure.

> Operand1 Operand2 Operator

Example

Operand1   Operand2   Operator

ab+

C Expressions

Prefix Expression

The expression in which the operator is used before operands is called a prefix expression.

The prefix expression has the following general structure.

> Operator Operand1 Operand2

Example

Operator   Operand1   Operand2

$$+ab$$

C Operator Precedence and Associativity

What is Operator Precedence?

Operator precedence is used to determine the order of operators evaluated in an expression. In c programming language every operator has precedence (priority). When there is more than one operator in an expression the operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.

What is Operator Associativity?

Operator associativity is used to determine the order of operators with equal precedence evaluated in an expression. In the c programming language, when an expression contains multiple operators with equal precedence, we use associativity to determine the order of evaluation of those operators.

C Operator Precedence and Associativity

Let's understand the precedence by the example given below:

```c
int value=10+20*10;
```

The value variable will contain **210** because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C operators is given below:

Category	Operator	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

Category	Operator	Associativity
Equality	== !=	Left to right
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

In the above table, the operator precedence decreases from top to bottom and increases from bottom to top.

Operators and Expressions/Dr.B.Vinodhini/ASP/CSE/SNSCT