SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

19ITT101-PROGRAMMING IN C AND DATA STRUCTURES

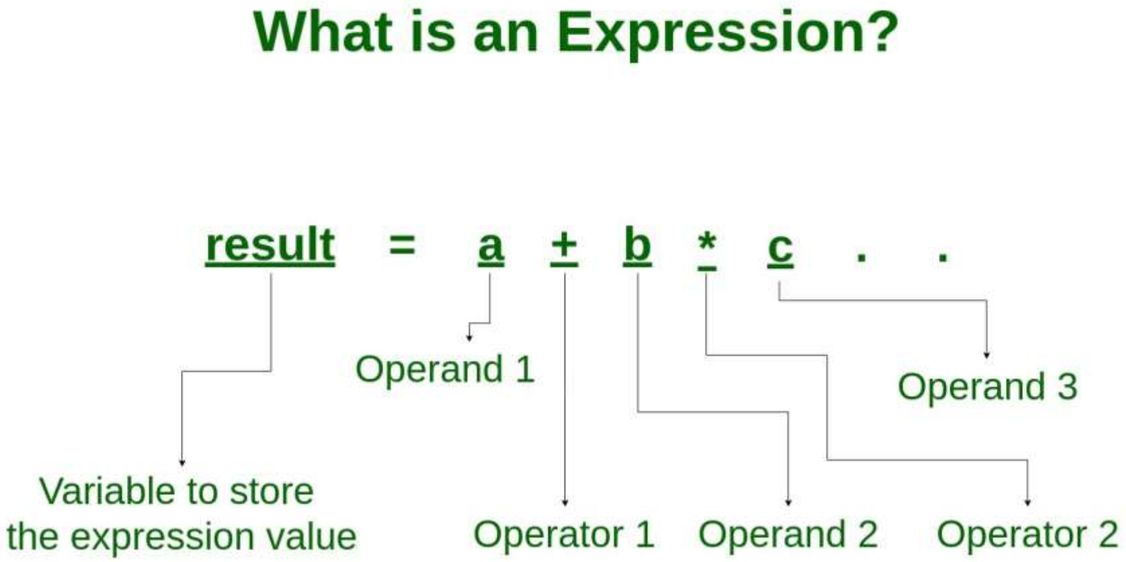
I YEAR - II SEM

UNIT 1 – INTRODUCTION TO C

TOPIC 8 – Operators & Expressions

C OPERATORS & EXPRESSIONS

- An **operator** is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- Operators are used in programs to manipulate data and variables.
- They usually form a part of the mathematical or logical **expressions**.
- An **expression** is a sequence of ‘operands’ and ‘operators’ that reduces to a single value.
- For example, 10 + 15 is an expression whose value is 25.



Algebric Expression	C Expression
$a \times b - c \times d$	a * b – c * d
$(m + n)\ (a + b)$	(m + n) * (a + b)
$3x2 + 2x + 5$	3 * x * x + 2 * x + 5
$\frac{a+b+c}{d+e}$	(a + b + c) / (d + e)
$\left[\frac{2BY}{d+1} - \frac{x}{3(z+y)}\right]$	2 * b * y / (d + 1) – x / 3 * (z + y)

C OPERATOR CLASSIFICATION

C operators can be classified into a number of categories. They include:

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

1. ARITHMETIC OPERATORS

- C provides all the basic arithmetic operators.
- The operators **+, –, *, /** all work the same way as they do in other languages.
- These can operate on any built-in data type allowed in C.

Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1
+ +	Increment	A = 10; A+ +	11
– –	Decrement	A = 10; A– –	9

INTEGER ARITHMETIC

- When both the **operands** in a single arithmetic expression such as a+b are integers, the expression is called an integer expression, and the ‘operation’ is called **integer arithmetic**.

- Integer arithmetic always yields an integer value.

- Example, if a and b are integers, then for **a = 14** and **b = 4** we have the following results:
  - a – b = 10
  - a + b = 18
  - a * b = 56
  - a / b = 3 (decimal part truncated)
  - a % b = 2 (remainder of division)

REAL ARITHMETIC

- An arithmetic operation involving **only real operands** is called real arithmetic.

- A real operand may assume values either in **decimal or exponential** notation.

- Since floating point values are rounded to the number of significant digits permissible, the final value is an approximation of the correct result.

- Example, If x, y, and z are floats, then we will have:
  - x = 6.0/7.0 = 0.857143
  - y = 1.0/3.0 = 0.333333
  - z = –2.0/3.0 = –0.666667

- The operator % cannot be used with real operands.

MIXED-MODE ARITHMETIC

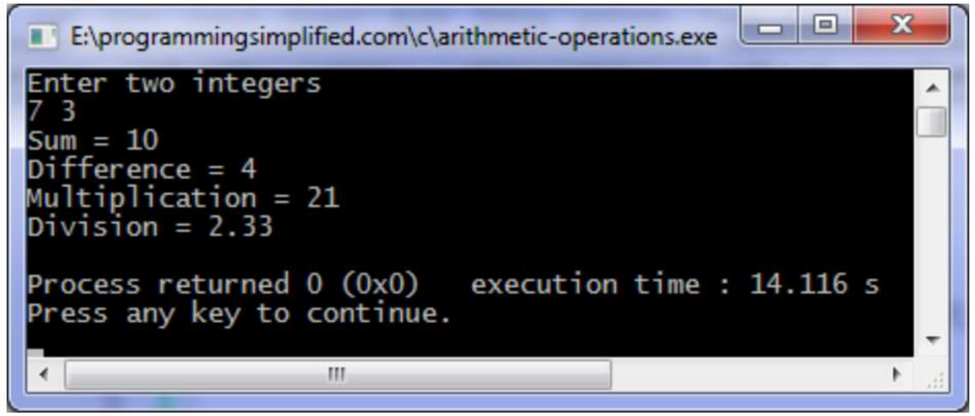
- When one of the operands is ‘real’ and the other is ‘integer’, the expression is called a mixed-mode arithmetic expression.
- If either operand is of the real type, then only the real operation is performed and the result is always a real number.

Operation	Result	Example
Int/int	Int	2/5 = 0
Real/int	Real	5.0/2 = 2.5
Int/real	Real	5/2.0 = 2.5
Real/real	real	5.0/2.0 = 2.5

C PROGRAM FOR ARITHMETIC OPERATIONS

```c
#include <stdio.h>

int main()
{
   int first, second, add, subtract, multiply;
   float divide;

   printf("Enter two integers\n");
   scanf("%d%d", &first, &second);

   add = first + second;
   subtract = first - second;
   multiply = first * second;
   divide = first / (float)second;   //typecasting, you can also write: divide = (float)first/second

   printf("Sum = %d\n", add);
   printf("Difference = %d\n", subtract);
   printf("Multiplication = %d\n", multiply);
   printf("Division = %.2f\n", divide); // "%.2lf" to print two decimal digits, by default (%lf) we get six

   return 0;
}
```

Output of program:



2. RELATIONAL OPERATORS

- We often **compare two quantities** and depending on their relation, take certain decisions.
- For example, we may compare the age of two persons, or the price of two items, and so on.
- These comparisons can be done with the help of relational operators.
- The symbol '<', meaning 'Less Than' and '>', meaning 'Greater Than'.
- An expression such as a < b or 1 < 20 containing a relational operator is termed as a relational expression.
- The value of a relational expression is either **one or zero**.
- Value = 1, if the specified relation is True
- Value = 0, if the specified relation is False
- For example
  - 10 < 20 is true → Value = 1
  - 20 < 10 is false → Value = 0
- Relational expressions are used in decision statements such as **if** and **while** to decide the course of action of a running program.

2. RELATIONAL OPERATORS

ae-1 relational operator ae-2

» Example, 10 < 20

Relational Operators

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

3. LOGICAL OPERATORS

C has the following three logical operators.

1. **&&** meaning logical AND
2. **||** meaning logical OR
3. **!** meaning logical NOT

- The logical operators && and || are used when we want to test more than one condition and make decisions.
- Example : a > b && x == 10
- An expression of this kind, which combines two or more **relational expressions**, is termed as a logical
- expression or a compound relational expression.
- Like the simple relational expressions, a logical expression also yields a value of **one or zero**, according to the truth table shown.
- The logical expression given above is true only if a > b is true and x == 10 is true. If either (or both) of them are false, the expression is false.

Truth Table

op-1	op-2	Value of the expression	
		op-1 && op-2	op-1 || op-2
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

4. ASSIGNMENT OPERATORS

- Assignment operators are used to assign the result of an expression to a variable.
- The usual assignment operator is, '='.

- In addition, C has a set of '**shorthand** ' assignment operators of the form

  v op= exp;
- Where **v** is a variable, **exp** is an expression and **op** is a C binary arithmetic operator.
- The operator **op=** is known as the shorthand assignment operator.
- The assignment statement v op= exp; is equivalent to

  v = v op (exp);

- Example: x += y+1;
- This is same as the statement x = x + (y+1);
- The shorthand operator **+=** means 'add y+1 to x' or 'increment x by y+1'.
- For y = 2, the above statement becomes x += 3; and when this statement is executed, 3 is added to x.
- If the old value of x is, say 5, then the new value of x is 8.

4. ASSIGNMENT OPERATORS

- The use of shorthand assignment operators has three advantages:
  1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
  2. The statement is more concise and easier to read.
  3. The statement is more efficient.

Shorthand Assignment Operators

Statement with simple assignment operator	Statement with shorthand operator
a = a + 1	a += 1
a = a – 1	a –= 1
a = a * (n+1)	a *= n+1
a = a / (n+1)	a /= n+1
a = a % b	a %= b

5. INCREMENT AND DECREMENT OPERATORS

- C allows two very useful operators not generally found in other languages.
- These are the **increment and decrement** operators:

  ++ and — –

- The operator ++ adds 1 to the operand, while – – subtracts 1.

- Both are unary operators and takes the following form:

  ++m; or m++;

  – —m; or m– —;
- ++m; is equivalent to m = m+1; (or m += 1;)
- – —m; is equivalent to m = m–1; (or m –= 1;)

- We use the increment and decrement statements in **for** and **while** loops extensively.

5. INCREMENT AND DECREMENT OPERATORS

- While ++m and m++ mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement.

- Consider the following:

  m = 5;

  y = ++m;

- In this case, the value of y and m would be **6**.
- Suppose, if we rewrite the above statements as

  m = 5;

  y = m++;

- then, the value of y would be **5** and m would be **6**.

- M++;- Post increment, first do the operation and then increment
- ++m;- Pre increment, first increment and then do the operation
- --m; - Pre decrement, first decrement and then do the operation
- M--; - Post decrement, first do the operation and then increment

- A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left.
- On the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

6. CONDITIONAL OPERATOR

- A ternary operator pair "? :" is available in C to construct conditional expressions of the form

  exp1 ? exp2 : exp3

- where exp1, exp2, and exp3 are expressions.
- The operator ? : works as follows:

  exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of the expression.

  If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.

  Note that only one of the expressions (either exp2 or exp3) is evaluated.
- For example, consider the following statements:

```
a = 10;
b = 15;
x = (a > b) ? a : b;
```

- In this example, x will be assigned the value of b.
- This can be achieved using the if..else statements as follows:

```
if (a > b)
        x = a;
else
        x = b;
```

7. BITWISE OPERATORS

- C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level.
- These operators are used for testing the bits, or shifting them right or left. Bitwise operators may not be applied to float or double.

Bitwise Operators

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right

8. SPECIAL OPERATORS

- C supports some special operators of interest such as
- Comma operator (,)
- Sizeof operator (sizeof)
- Pointer operators (& and *)
- Member selection operators (. and –>).

- **The Comma Operator**
- The comma operator can be used to link the related expressions together.
- A comma-linked list of expressions are evaluated **left to right** and the value of right-most expression is the value of the combined expression.
- For example, the statement

  value = (x = 10, y = 5, x+y);
- first assigns the value 10 to x
- then assigns 5 to y
- and finally assigns 15 (i.e. 10 + 5) to value.
- Since comma operator has the lowest precedence of all operators, the **parentheses** are necessary.

8. SPECIAL OPERATORS

The sizeof Operator

- The sizeof is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies.
- The operand may be a variable, a constant or a data type qualifier.
- Examples

  m = sizeof (sum);

  n = sizeof (long int);
- The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer.
- It is also used to allocate memory space dynamically to variables during execution of a program.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
        Int a;
        Printf("size of variable a is : %d",sizeof(a));
}
```

OUTPUT:

size of variable a is : 2

8. SPECIAL OPERATORS

- **Pointer Operator**
- &
- This symbol specifies the address of the variable

- *
- This symbol specifies the value of the variable.

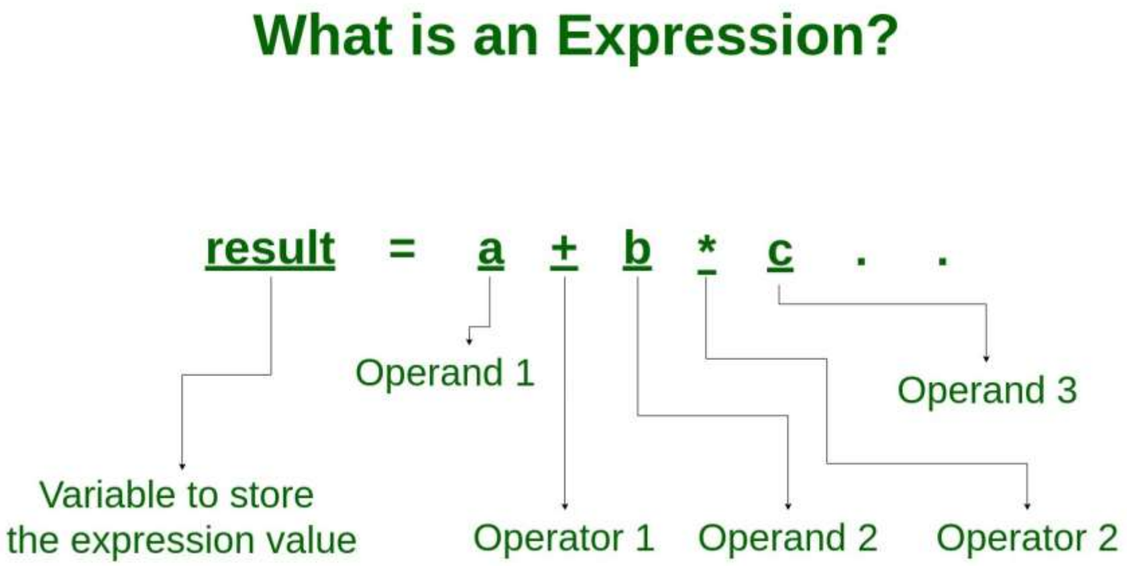
- **Member Selection Operator**
- . and - - >
- Used to access the elements from a structure.

ARITHMETIC EXPRESSIONS

- An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language.

  Variable = Expression;

- Whenever this statement is encountered, the expression is evaluated first and the result then replaces the previous value of the variable on the left-hand side.
- All variables used in the expression must be assigned values before evaluation is attempted.



Algebric Expression	C Expression
$a \times b - c \times d$	a * b – c * d
$(m + n)\ (a + b)$	(m + n) * (a + b)
$3x2 + 2x + 5$	3 * x * x + 2 * x + 5
$\frac{a+b+c}{d+e}$	(a + b + c) / (d + e)
$\left[\frac{2BY}{d+1} - \frac{x}{3(z+y)}\right]$	2 * b * y / (d + 1) – x / 3 * (z + y)

PRECEDENCE OF ARITHMETIC OPERATORS

- An arithmetic expression without parentheses will be evaluated from **left to right** using the rules of precedence of operators.
- There are two distinct priority levels of arithmetic operators in C:
  - High priority * / %
  - Low priority + –
- The basic evaluation procedure includes '**two' left-to-right passes** through the expression.
- During the first pass, the **high priority operators** (if any) are applied as they are encountered.
- During the second pass, the **low priority operators** (if any) are applied as they are encountered.
- Consider the following evaluation.

  x = a–b/3 + c*2–1
- When a = 9, b = 12, and c = 3, the statement becomes

  x = 9–12/3 + 3*2–1
- and is evaluated as follows
- First pass:
  - Step1: x = 9– 4+3*2–1
  - Step2: x = 9– 4+6–1
- Second pass
  - Step3: x = 5+6–1
  - Step4: x = 11–1
  - Step5: x = 10

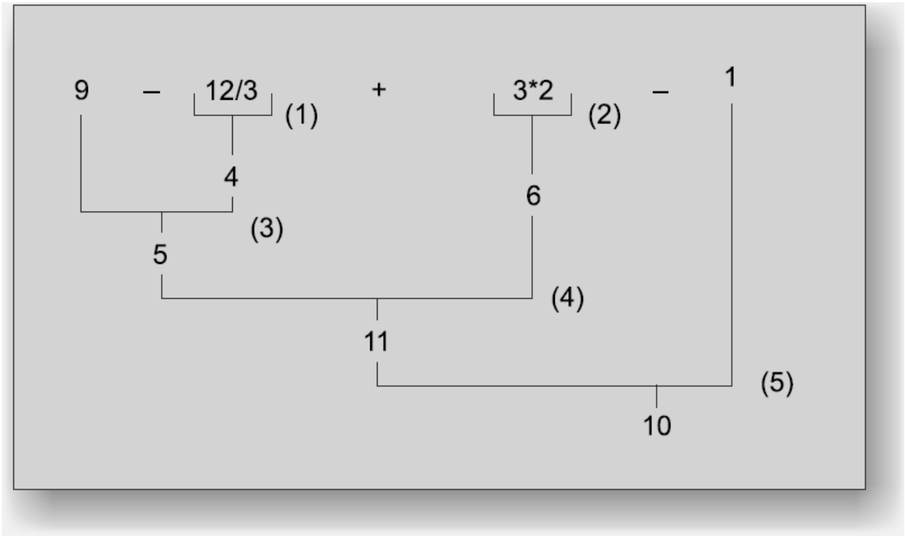


Illustration of hierarchy of operations

PRECEDENCE OF ARITHMETIC OPERATORS

- The order of evaluation can be changed by introducing parentheses into an expression.
- Consider the same expression with parentheses as shown below:

  9–12/(3+3)*(2–1)
- Whenever **parentheses** are used, the expressions within parentheses assume highest priority.
- If two or more sets of parentheses appear one after another as shown above, the expression contained in the left-most set is evaluated first and the right-most in the last.
- Given below are the new steps.
- First pass
  - Step1: 9-12/6 * (2-1)
  - Step2: 9-12/6 * 1
- Second pass
  - Step3: 9-2 * 1
  - Step4: 9-2
- Third pass
  - Step5: 7

- Parentheses may be nested, and in such cases, evaluation of the expression will proceed outward from the **innermost** set of parentheses.

RULES FOR EVALUATION OF EXPRESSION

- First, parenthesized sub expression from **left to right** are evaluated.
- If parentheses are **nested,** the evaluation begins with **the innermost sub-expression**
- The precedence rule is applied in determining the order of application of operators in evaluating **sub-expressions.**
- The **associativity rule** is applied when two or more operators of the same precedence level appear in a sub-expression.
- Arithmetic expressions are evaluated from left to right using the rules of precedence.
- When parentheses are used, **the expressions within parentheses** assume highest priority.

OPERATOR PRECEDENCE AND ASSOCIATIVITY

- Each operator, in C has a precedence associated with it.
- This precedence is used to determine how an expression involving more than one operator is evaluated.
- There are **distinct levels of precedence** and an operator may belong to one of these levels.
- The operators at the **higher level of precedence are evaluated first**.
- The operators of the same precedence are evaluated either from ‘left to right’ or from ‘right to left’, depending on the level.
- This is known as the associativity property of an operator.
- **Table below** provides a complete list of operators, their precedence levels, and their rules of association.
- The groups are listed in the order of **decreasing precedence**.
- Rank 1 indicates the highest precedence level and 15 the lowest.

- <u>Rules of Precedence and Associativity</u>
  - Precedence rules decide the order in which different operators are applied
  - **Associativity rule** decides the order in which multiple occurrences of the same level operator are applied.

OPERATOR PRECEDENCE AND ASSOCIATIVITY

Operator	Description	Associativity	Rank
( )	Function call	Left to right	1
[ ]	Aray element reference		
+	Unary plus		
–	Unary minus	Right to left	2
++	Increment		
– –	Decrement		
!	Logical negation		
~	Ones complement		
*	Pointer reference (indirection)		
&	Address		
sizeof	Size of an object		
(type)	Type cast (conversion)		
*	Multiplication	Left to right	3
/	Division		
%	Modulus		
+	Addition	Left to right	4
–	Subtraction		
<<	Left shift	Left to right	5
>>	Right shift		
<	Less than	Left to right	6
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
==	Equality	Left to right	7
!=	Inequality		
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional expression	Right to left	13
=	Assignment operators	Right to left	14
* = /= %=			
+= –= &=			
^= |=			
<<= >>=			
,	Comma operator	Left to right	15