SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

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 ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23ITT101- PROBLEM SOLVING & C PROGRAMMING

I YEAR /I SEMESTER

Unit II – C PROGRAMMING BASICS

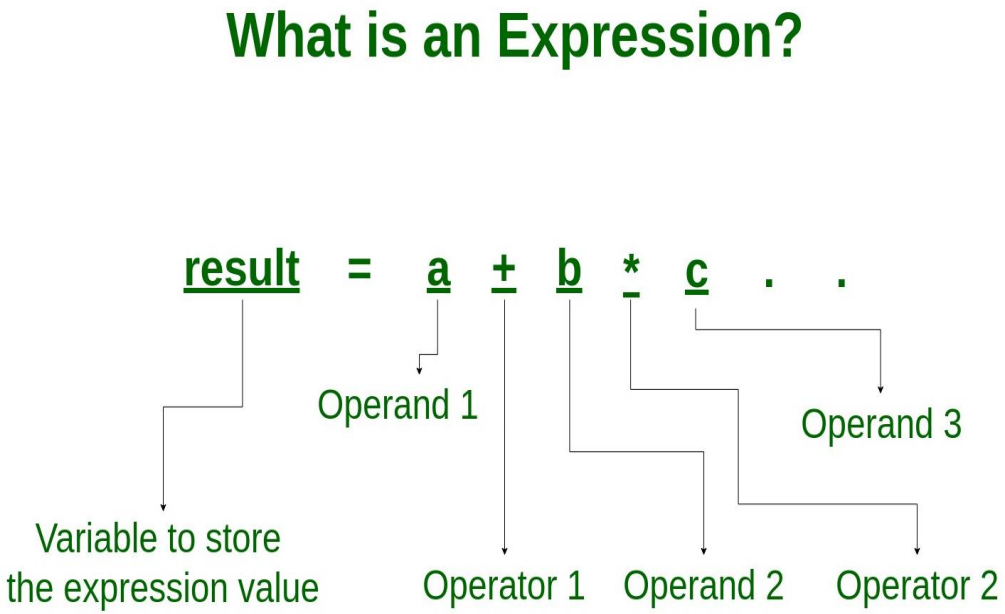
Topic : Expressions

Topics to be covered

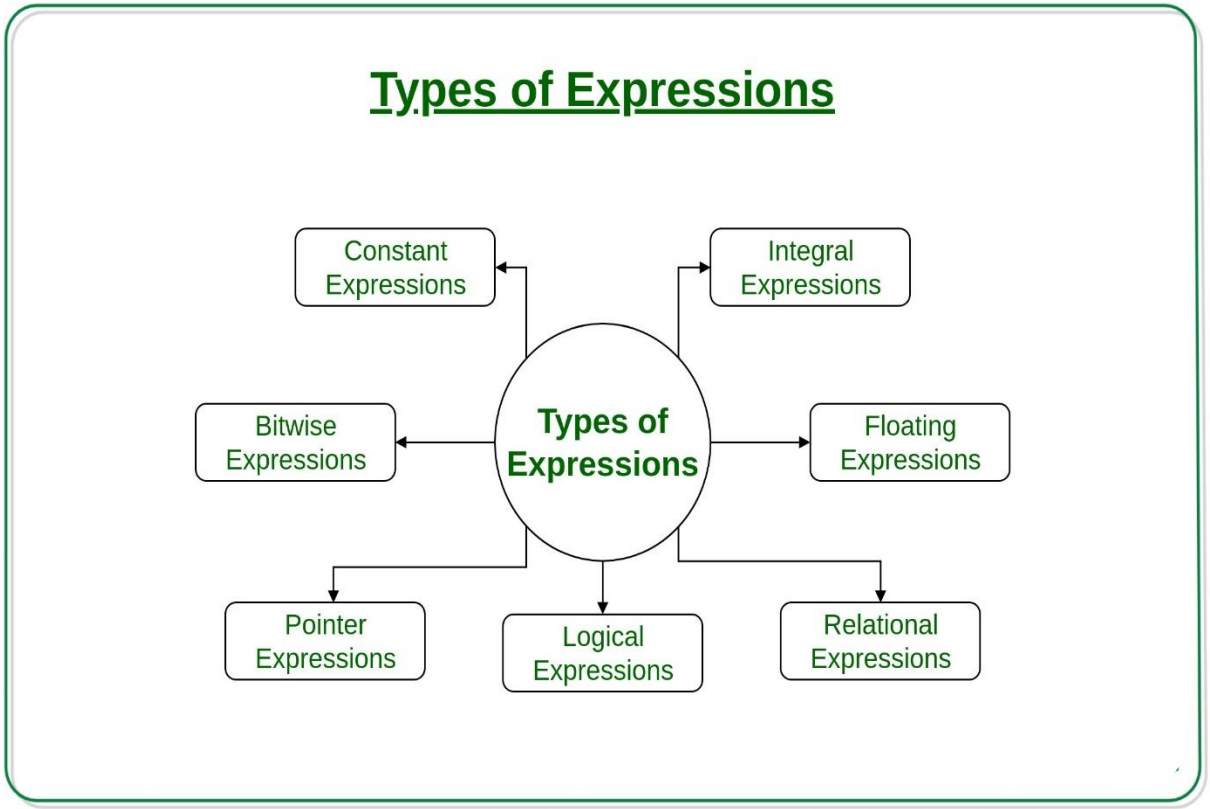
- Types of Expressions
- Evaluation of expressions
- Operator precedence and associativity
- Type conversion in expression

Expressions

An expression is a combination of operators, constants and variables. An expression may consist of one or more operands, and zero or more operators to produce a value.



Types of Expression



Constant expressions: Constant Expressions consists of only constant values. A constant value is one that doesn't change.

Examples: 5, 10 + 5 / 6.0, 'x'

Integral expressions: Integral Expressions are those which produce integer results after implementing all the automatic and explicit type conversions.

Examples: x, x * y, x + int(5.0)

where x and y are integer variables.

Floating expressions: Float Expressions are which produce floating point results after implementing all the automatic and explicit type conversions.

Examples: x + y, 10.75

where x and y are floating point variables.

Relational or Boolean expressions: Relational Expressions yield results of type bool which takes a value true or false.

Examples: x <= y, x + y > 2

Logical expressions: Logical Expressions combine two or more relational expressions and produces bool type results.

Examples: x > y && x == 10, x == 10 || y == 5

Pointer expressions: Pointer Expressions produce address values.

Examples: &x, ptr, ptr++

where x is a variable and ptr is a pointer.

Bitwise expressions: Bitwise Expressions are used to manipulate data at bit level. They are basically used for testing or shifting bits.

Examples: x << 3 shifts three bit position to left

y >> 1 shifts one bit position to right.

Shift operators are often used for multiplication and division by powers of two.

Note: An expression may also use combinations of the above expressions can be termed as compound expressions.

Arithmetic Expression

- An arithmetic expression is an expression that consists of operands and arithmetic operators.
- pure integer expression : expression that contains only integral operands.
- pure real expression : expression that contains only real operands
- mixed mode expression: expression that contains both integral and real operands.

Forms of Arithmetic Expressions

Expression Types (based on the operator position)

- Infix Expression - Operator is used between operands is

  Syntax: Operand1 Operator Operand2

  Example : c=a+b;

- Postfix Expression - Operator is used after operands

  Syntax: Operand1 Operand2 Operator

  Example : c=ab+

- Prefix Expression - Operator is used before operands

  Syntax: Operator Operand1 Operand2

  Example : c=+ab;

Evaluation of Expression

- When there are multiple operators in an expression, they are evaluated according to operator precedence and associativity.
- The operator with higher precedence is evaluated first and the operator with least precedence is evaluated last.

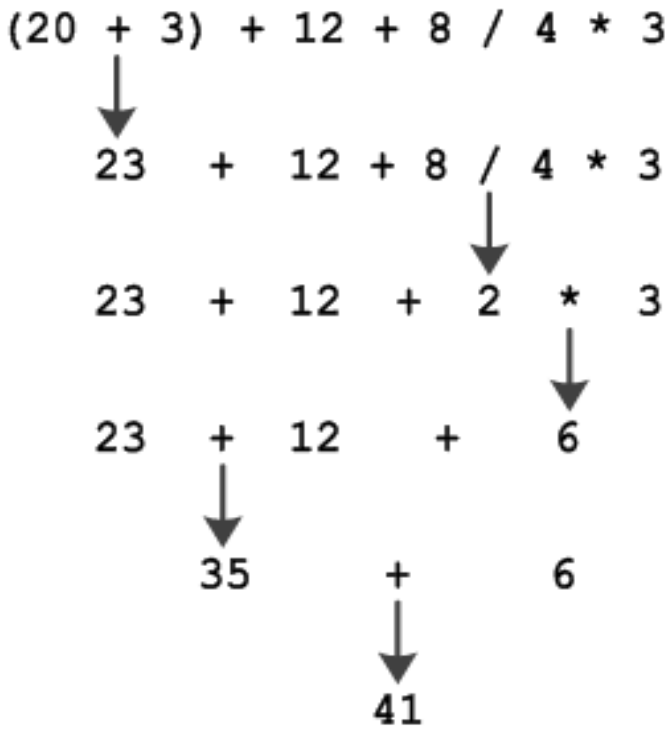
Operator Precedence

Operator precedence determines which operator is performed first in an expression with more than one operators with different precedence.

For example: Solve

10 + 20 * 30	is calculated as	10 + (20 * 30)
	and not as	(10 + 20) * 30

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the *innermost* pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they’re evaluated left to right.
*/%	Multiplication Division Remainder	Evaluated second. If there are several, they’re evaluated left to right.
+-	AdditionSubtraction	Evaluated third. If there are several, they’re evaluated left to right.
=	Assignment	Evaluated last.


(20 + 3) + 12 + 8 / 4 * 3
23 + 12 + 8 / 4 * 3
23 + 12 + 2 * 3
23 + 12 + 6
35 + 6
41

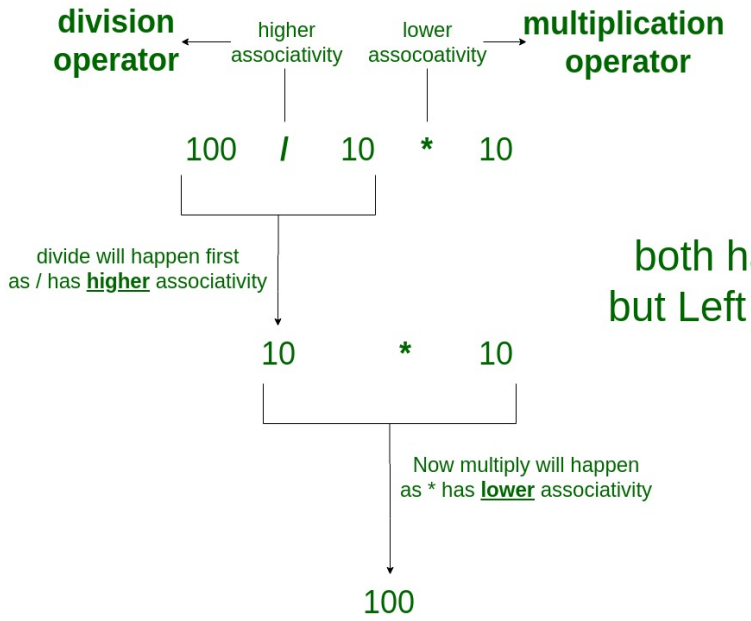
- Operators Associativity is used when two operators of same precedence appear in an expression.
- Associativity can be either Left to Right or Right to Left.

For example:

‘*’ and ‘/’ have same precedence and their associativity is Left to Right, so the expression “100 / 10 * 10” is treated as “(100 / 10) * 10”.

- Precedence and associativity of postfix ++ and prefix ++ are different
- Comma has the least precedence among all operators and should be used carefully
- There is no chaining of comparison operators in C

Operator Associativity



/ and ***
both have the same precedence
but Left to Right (**LTR**) associativity

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
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

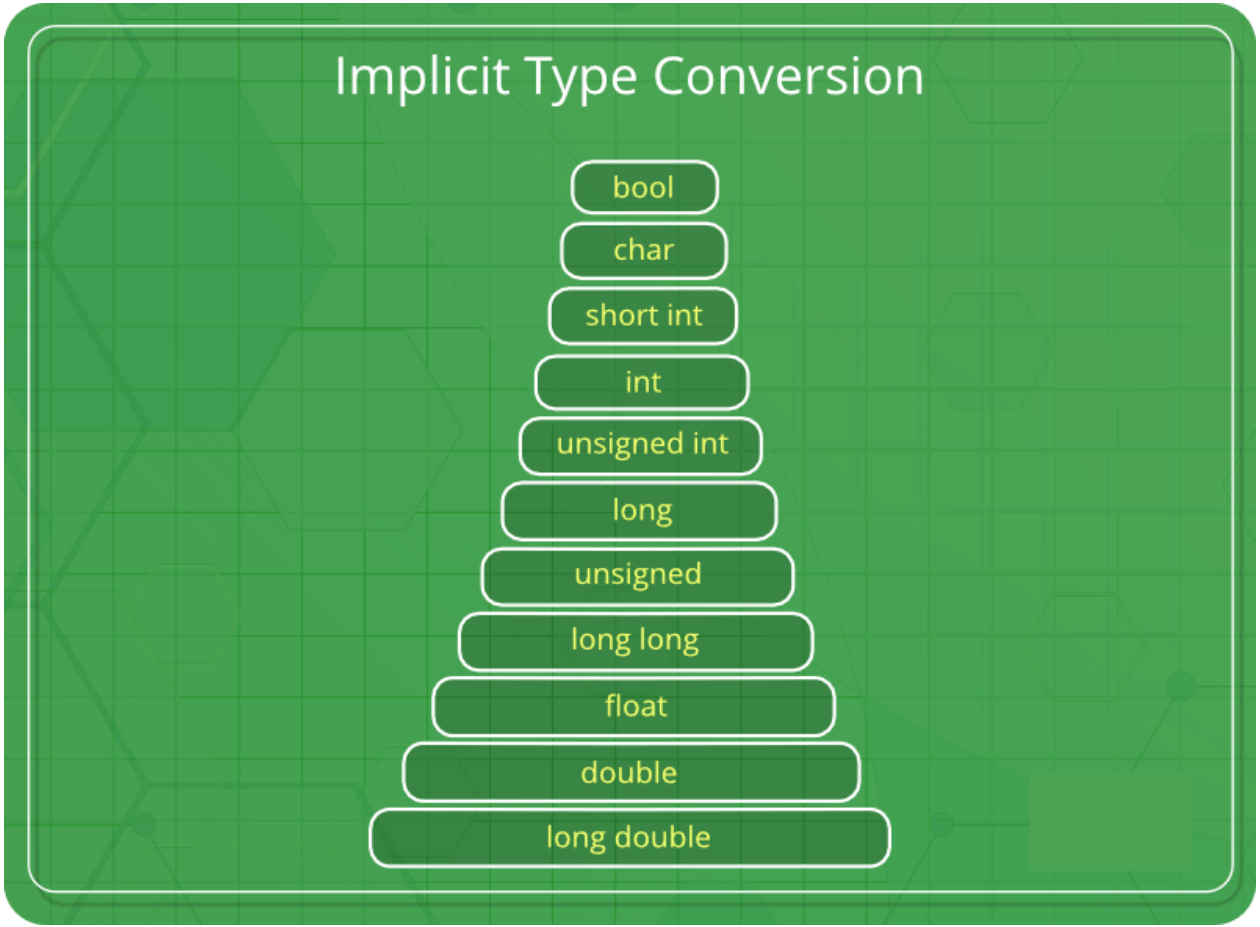
Type Conversion in Expression

A type cast is basically a conversion from one type to another.

There are two types of type conversion:

- Implicit or Automatic Type Conversion
- Explicit conversion

Implicit Type Conversion



- Done by the compiler on its own, without any external trigger from the user.
- Generally takes place when in an expression more than one data type is present.
- In such condition type conversion (type promotion) takes place to avoid lose of data.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.

  bool -> char -> short int -> int ->

  unsigned int -> long -> unsigned ->

  long long -> float -> double -> long double

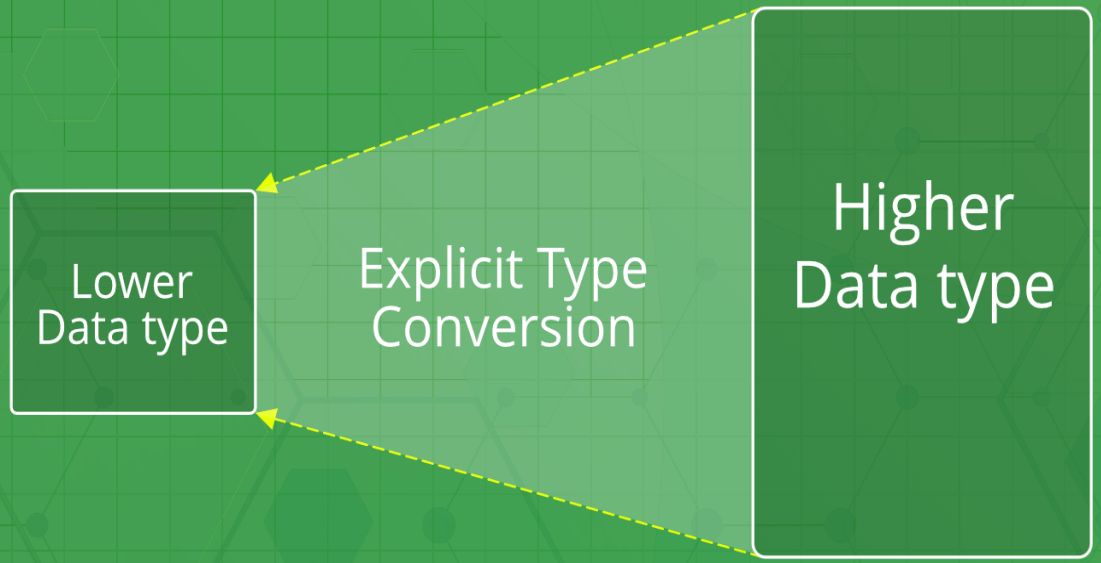
It is possible for implicit conversions

- to lose information
- signs can be lost (when signed is implicitly converted to unsigned)
- overflow can occur (when long long is implicitly converted to float)

```
// An example of implicit conversion
#include<stdio.h>
int main()
{
    int x = 10;    // integer x
    char y = 'a';  // character c

    // y implicitly converted to int. ASCII  value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

Output : x = 107, z = 108.000000

Explicit Type Conversion



- The process is also called type casting and it is user defined.
- Here the user can type cast the result to make it of a particular data type.

The syntax in C:

(type) expression

```
// C program to demonstrate explicit type casting
#include<stdio.h>
 int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    printf("sum = %d", sum);

    return 0;
}
```

Output: sum = 2

Advantages of Type Conversion

- This is done to take advantage of certain features of type hierarchies or type representations.
- It helps us to compute expressions containing variables of different data types.

Summary

- An expression is a formula in which operands are linked to each other by the use of operators to compute a value. An operand can be a function reference, a variable, an array element or a constant.


Thank
you