3. EXPRESSIONS

It is a sequence of operands and operators that reduce to a single value.

Operator : It is a symbolic token that represents an action to be taken.

Ex: * is an multiplication operator.

Operand: An operand receives operators action.

Ex: A + B

In this A and B are operands and + is an operator.

Note: Expressions always evaluate to a single value. There is no limit to the number of operator and operand sets in an expression.

Ex: 5.0/9.0*(fah –32.0)

EXPRESSION

An expression may be a simple constant or a simple name or a constant or name connected suitably by the help of an operator or it may be a parenthetical expression.

Example:

1. Integer Constant	5
2. Character Constant	‘a’
3. Name (variable)	sum
4. Constant and constant	5+6
5. Constant with variable	5 + sum
6. Variable with constant	sum + 5

In the above example case 4,5,6 are binary expressions, since there are two operands and one operator.

Arithmetic Operators

Integers, floating point numbers, and double precision numbers can be added, subtracted, multiplied or divided using operators called arithmetic operators.

Operation	Operator	Example
Addition	+	2+3 = 5, 2.3+3 = 5.3 2.3+5.2= 7.5
Subtraction	-	3-2 = 1, 3-2.3= 0.7 3.0-2.3=0.7
Multiplication	*	3*5=15, 3.1*5= 15.5 3.1*5.0=15.5
Division	/	5/2 =2, 4/3=1 5.0/2=2.5 5.0/2.0=2.5
Remainder	%	5%2=1 11%6=5

✓An expression contains only integer operands is called an integer expression.

✓An expression containing only floating-point operands is called a floating-point expression; the result is a double precision value.

✓Although it is usually better not to mix integer and floating-point operands in an arithmetic operation, the data type of each operation is determined by the following rule:

✓If all the operands are integer then the result is integer.

✓If any operand is floating point or double precision then the result is double.

INTEGER DIVISION

Dividing an integer by another integer will produce a strange result. The answer will not have a decimal point. The result will be an integer. The fractional part will be truncated.

Example

6/2 = 3 6/5 = 1 6/3 = 2 6/4 = 1

To capture the remainder after the integer division Remainder operator (%) can be used.

Example

6%2 = 0 6%5 = 1 6%3 = 0 6%4 = 2

Note: In remainder operation both the integers must be integer.

A Unary operator

The minus sign is a Unary operator. A minus sign place before a single numerical operand negates the number.

Example:

-4

-2.34567

-sum

Rules for writing expressions with more than one operators:

1. Two binary arithmetic operator symbols never be placed side by side.

Example : 4+ *2 is invalid because two operators + * are placed next to each other.

2. Parenthesis may be used to form groupings, and all the expressions enclosed within the parenthesis will be evaluated first.

3.When parenthesis are used within parenthesis, the expression in the inner most parenthesis will be evaluated first. The evaluation continues from innermost to outermost parenthesis until the expressions in all parentheses have been evaluated.

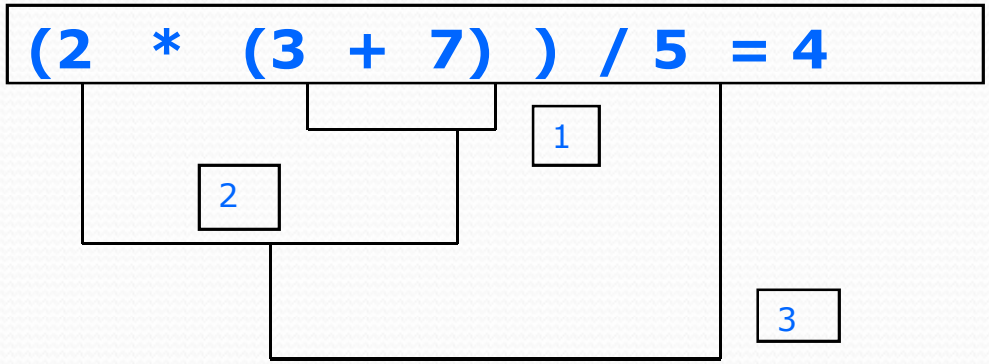
4. The number of right-facing parenthesis must be equal to the number of left-facing parenthesis.

Example:

(6+4) / (2+3) = 2

In the above example first (6+4) will be evaluated first and then (2+3) will be evaluated yielding the result as 10/5. The 10/5 will be evaluated to yield the result as 2.

Example:



5. Parenthesis can not be used to indicate multiplication. The multiplication operator * must be used.

Example:

(3+4) (4+5) is invalid

(3+4) * (4+5) is valid

As a general rule parenthesis is used to specify logical grouping of operands.

Postfix increment / decrement Operator

It consist of one operand followed be an operator.

The operators are :

++

– –

The general format of postfix operation is

Operand	Unary operator

Example: a++ sum++

a-- value—

The value of the expression is equal to the value of ‘a’ before the increment .

The meaning of ‘a++’ is increment the content of ‘a’ by one and store the result in ‘a’ itself.

The postfix operator has associativity from left to right.

Example :

Suppose a = 10; The value of the expression 'a++' is 10.

Then 'a++' will make the value of of a as: 10+1 = 11.

Example:

Suppose a = 10; The value of the expression 'a--' is 10.

Then 'a--' will make the value of a as: 10 –1 = 9;

```
/* program in c to implement ++ operator */
#include <stdio.h>
int main(void)
{
  int a;
  a = 10;
 printf("The value of a is %d \n",a);
 printf("The value of a++ is %d\n", a++);
 printf("The new value of a is %d\n",a);
return 0;
}
```

Results:

```
10
10
11
```

Prefix increment / decrement Operator

It consist of one operator followed be an operand

The operators are :

++

– –

The general format of postfix operation is

Unary operator	Operand

Example:

++a

++sum

--a

--value

The value of the expression is equal to the value of a + 1. The meaning of '++a' is increment the content of 'a' by one and store the result in 'a' itself. The prefix operator will have association from right to left.

Example :

Suppose a = 10; The value of the expression '++a' is 11.
Then '++a' will make the value of of a as: 10+1 = 11.

Example:

Suppose a = 10; The value of the expression '- -a' is10.
Then '--a' will make the value of a as: 10 –1 = 9;

```
/* program in c to implement ++ operator */
#include <stdio.h>
int main(void)
{
  int a;
  a = 10;
 printf("The value of a is %d\n",a);
 printf("The value of ++a is %d\n", ++a);
 printf("The new value of a is %d\n",a);
return 0;
}
```

Results:

```
10
11
11
```

sizeof( ) operator

Though sizeof( ) looks like a function , it is an operator. The sizeof operator returns the number of bytes of the data type included in the parenthesis. The sizeof operator is an integral part of the C language.

```
#include<stdio.h>
int main(void)
{
printf("The size of char is d\n",sizeof(char));
printf("The size of int is %d\n",sizeof(int));
printf("The size of float is %d \n", sizeof(float));
printf("The size of double is %d\n", sizeof(double));
return 0;
}
```

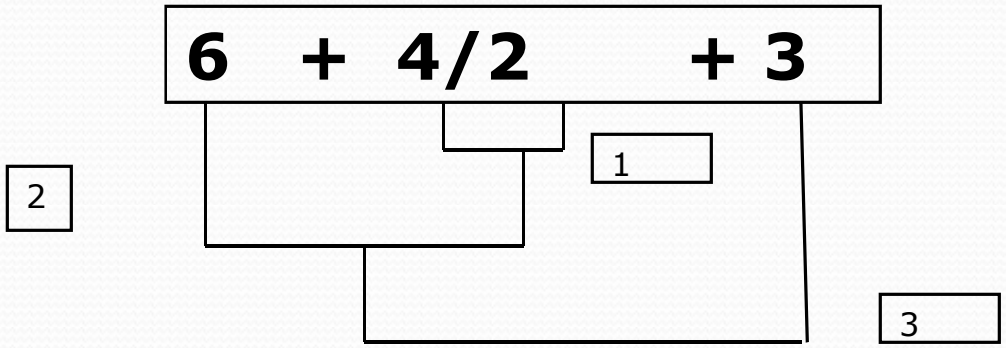
Result:

```
1
2
4
8
```

Operator Precedence

- Precedence of an operator establishes its priority relative to other operators.
- Precedence is used to determine the order in which different operators in a complex expression are evaluated.
- Each operator has a precedence value. The operator with a higher precedence number is used before the operators with lower precedence.

Example:



Precedence Table

Precedence	Operator	Description	Side Effect	Type	Associativity
18	( )	Identifier Constant Parenthetical expression	No	Primary	N/A
17	( ) [ ] -> .	Function Array Structure selection Structure member	Yes No No No	Postfix	Left
16	++ --	Postfix increment • decrement	Yes		
15	++ -- sizeof + - ! & * ~	Prefix increment • decrement Size of object in bytes Plus • Minus Not Address Indirection One's complement	Yes No No No No No No	Unary	Right
14	( )	Type cast	No		
13	* / %	Multiply • Divide • Modulus	No	Binary	Left
12	+ -	Addition • Subtraction	No		
11	<< >>	Bit Left • Bit Right	No		
10	< <= > >=	Comparison	No		
9	== !=	Equal • Not Equal	No		
8	&	Bit And	No		
7	^	Bit Exclusive Or	No		
6	|	Bit Inclusive Or	No		
5	&&	Logical And	No		
4	||	Logical Or	No		
3	? :	Conditional	No	Ternary	Right
2	= += -= *= /= %= >>= <<= &= &= ^= |=	Assignment Bit Assignment	Yes	Assignment	Right
1	,	Comma	No	Comma	Left

--a * (3 + b) /2 - c++ * b

If initially, a = 3, b = 4 and c = 5

2 * (3 + 4) / 2 - 5 * 4

2 * 7 / 2 - 5 * 4

14 / 2 - 5 * 4

7- 5 * 4

7- 20 = -13

Warning: If a single variable is modified more than once in an expression, the result is undefined.

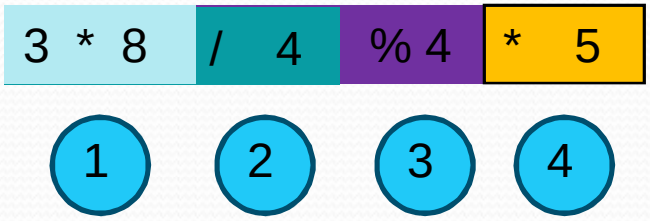
Associativity

- Associativity is used to determine the order in which the operators with the same precedence are evaluated in a complex expression.

- Precedence is applied before associativity to determine the order in which expressions are evaluated. Associativity is applied next.

- Associativity can be from the left or the right. Left associativity evaluates the expression from the left and moving to the right. The right associativity evaluates the expression by processing from the right to the left. Remember, associativity is only when the operators all have the same precedence.

3 * 8 / 4 % 4 * 5	= ((((3 * 8) / 4) % 4) * 5)



Right Associativity

- Only three types of expressions associate from the right. They are the unary expressions (Pre increment ++ and decrement --, +, -, !, address &, indirection *, one's complement ~), Conditional operator (? :) and assignment statement(=, +=, -=, *=, /=, %=, >>=, <<=, &=, |=, ^=).

- When more than one assignment operator occurs in an assignment expression, the assignment expression must be evaluated from the right to left.

a += b *= c -= 5

(a = a + (b = b * (c = c-5)))

If initially, a = 3, b = 5 and c = 8

(a = 3 + (b = 5 * (c = 8-5)))

(a = 3 + (b = 5 * (3)))

(a = 3 + (15))

a =18

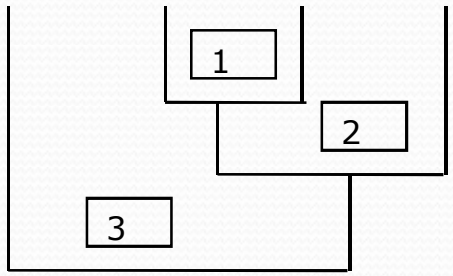
- Associativity is applied when we have more than one operator of the same precedence level in an expression.

- Expressions containing operators with the same precedence are evaluated according to their association.

- This means that evaluation is either from left to right or right to left as each operator is encountered.

Example:

7 + 2 * 5 / 3 = 10



Example:

If a = 101; b = 20; c = 30 Evaluate the following expression.

```
-- a  *  (1+b) / 3 - c++  *  b
```

```
100  * (1  + 20) / 3  - 30 * 20
```

```
=  100
```

Assignment Expression (Simple)

Variable	=	Expression

```
A  =  20
```

- The expression on the right hand side of the assignment operator will be evaluated first.

- The evaluated single value will be assigned to the variable name on the left hand side.

Example :

Let x = 20, y = 30. When the assignment expression z = x + y is evaluated the result will be z = 50;

Compound Assignment:

A compound assignment is a short hand notion for a simple assignment. The operators for the compound assignment are :

*=, /=, +=, -=, %=.

Compound Expression	**Meaning**
x+=y	x = x + y
x-=y	x = x - y
x*=y	x = x * y
x/=y	x = x / y
x%=y	x = x % y

The assignment operator has the lowest precedence level.
m *= x + 3 is equivalent to : m = m*(x+3)
Since x+3 will be evaluated first.

Mixed Type Expressions:

When the data types in an expression are of mixed type then expression will be evaluated according to the following promotion hierarchy.

Note: In an assignment expression, the final value must be same type as the left operand.

Explicit Type Conversion (cast Operator)

By explicitly specifying the type of data within ( ) data type conversion can be done.
The operand must be an unary expression.

Example

To convert a data from int to float,
use (float) a /* assuming a is of type int */
if x is of type float, (int) x gives a integer type value.

STATEMENTS

A statement is an action to be performed by the program. It translates into one or more executable computer instructions.

ANSI C supports six types of statements.

1.Expression Statement
2. Compound Statement
3. Labeled Statement
4. Selection Statement
5. Iterative Statement
6. Jump Statement

Expression Statement

When an expression is terminated by a semicolon then it is called an expression statement. When an expression statement is executed, C completes the pending side effects and discards the expression value before continuing with the next statement.

Example

```
x = 2;
```

The effect of the expression statement is to store the value of 2 to x. The value of the expression will be discarded after the value is stored in the variable.

Compound statement

A compound statement is a unit code consisting of zero or more statements. It is also known as a block. A compound statement consists of a an opening brace "{ " , an optional declartion, optional statement section, followed by a closing brace "}".

Example

```
{                                   <= Opening
Brace
        int x,y;
        x = 0;
        y = 0;
        prinyf("%d %d\n",x,y);
}                                   <= Closing
Brace
```

#define directive for creating constant macro

Syntax: #define NAME value

Examples:

```
#define PHI  22.0/7.0
#define MAX_LENGTH 100
#define MILES  0.8930
#define DEGREE_FACTOR 57.295779
```

Simple Programs

/*program in C to compute the area, circumference of a circle given the radius of a circle */

```
#include<stdio.h>
#define PHI 3.1412857
void main(void)
{
float radius,circum,area;
printf("Please input the radius ");
scanf("%f",&radius);
area = PHI*radius*radius;
circum = 2.0*PHI*radius;
printf("Radius of the circle is %f\n",radius);
printf("The area of the circle %f\n",area);
printf("The circumference is %f\n",circum);
return;
}
```

```
/*Program to find the sum of all the digits of a three digit number*/
#include<stdio.h>
void main(void)
{
 int number,fd,sd,td,sum;
printf("A three digit number please ");
scanf("%f",&number);
td = number – (number/10)*10;
number = number/10;
sd = number – (number/10)*10;
fd = number/10;
sum = fd+sd+td;
printf("The sum of the digits of the
                    number is %d\n",sum);
return;
}
 if number = 234, td = 234 – (234/10)*10 = 234 – 23*10 = 4
then number = number/10 = 234/10 = 23;
sd = 23 – (23/10)*10 = 23 – 20 = 3;
fd = 23/10 = 2; sum = 2+3+4 = 9;
```