CL-1002 Programming Fundamentals

LAB - 04

Introduction of operators and math.h library functions

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
Fall 2021

OPERATORS:

C Arithmetic Operators

There are many operators in C for manipulating data which include arithmetic Operators, Relational Operators, Logical operators and many more which will be discussed accordingly. Some of the fundamental operators are:

Operator	Description	Example (A=10, B=20)
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Example code:

```
// 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 Assignment Operators

Operator	Example	Equivalent Expression
+=	a+=b	a=a+b
−=	a-=b	a=a-b
*=	a*=b	a=a*b
/=	a/=b	a=a/b
%=	a%=b	a=a%b

C Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	`5 == 3` is evaluated to 0
>	Greater than	`5 > 3` is evaluated to 1
<	Less than	`5 < 3` is evaluated to 0
!=	Not equal to	`5 != 3` is evaluated to 1
>=	Greater than or equal to	`5 >= 3` is evaluated to 1
<=	Less than or equal to	`5 <= 3` is evaluated to 0

Example code:

```
#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;
}
```

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c==5) && (d>5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c==5) || (d>5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `!(c==5)` equals to 0.

Example code:

```
#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;
}
```

Bitwise Operators C

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power. Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

```
// C Program to demonstrate use of bitwise operators

#include <stdio.h>
int main()
{
      // a = 5(00000101), b = 9(00001001)
      unsigned char a = 5, b = 9;

      // The result is 00000001
      printf("a = %d, b = %d\n", a, b);
      printf("a&b = %d\n", a & b);

      // The result is 00001101
      printf("a|b = %d\n", a | b);

      // The result is 00001100
      printf("a^b = %d\n", a ^ b);

      // The result is 11111010
      printf("~a = %d\n", a = ~a);

      // The result is 00010010
      printf("b<<1 = %d\n", b << 1);

      // The result is 00000100
      printf("b>>1 = %d\n", b >> 1);
      return 0;
}
Output: a = 5, b = 9
a&b = 1
a|b = 13
a^b = 12
~a = 250
b<<1 = 18
b>>1 = 4
```

Comma Operator

Comma operators are used to link related expressions together. For example:

int a, c = 5, d;

The sizeof operator

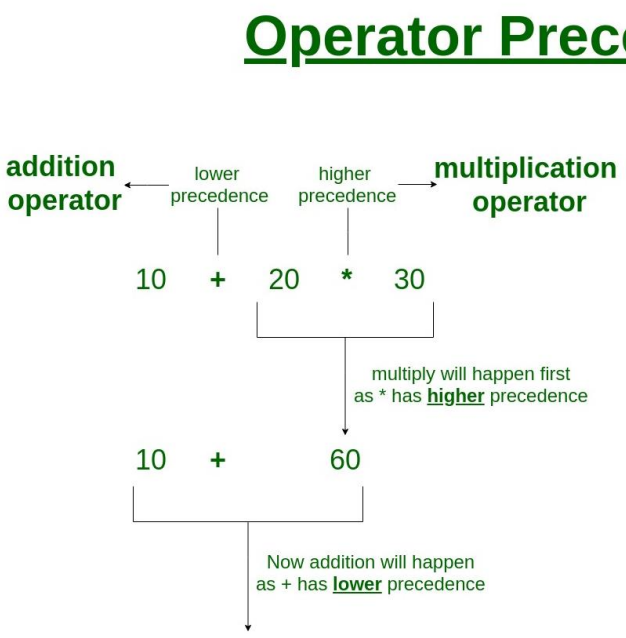
The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

Example code:

```
include <stdio.h>
int main()
{
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=%lu bytes\n",sizeof(a));
    printf("Size of float=%lu bytes\n",sizeof(b));
    printf("Size of double=%lu bytes\n",sizeof(c));
    printf("Size of char=%lu byte\n",sizeof(d));

    return 0;
}
```

Operators Precedence in C



Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.
Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

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

REFERENCE SITE:
https://www.geeksforgeeks.org/operator-precedence-and-associativity-in-c/

Example code:

```
#include <stdio.h>

main() {

   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a + b) * c / d;       // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n",  e );
```

```
   e = ((a + b) * c) / d;     // (30 * 15 ) / 5
   printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );

   e = (a + b) * (c / d);   // (30) * (15/5)
   printf("Value of (a + b) * (c / d) is  : %d\n",  e );

   e = a + (b * c) / d;      //  20 + (150/5)
   printf("Value of a + (b * c) / d is  : %d\n" ,  e );

   return 0;
}
```

When compile and execute the above program, it produces the following result :

Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is : 90
Value of (a + b) * (c / d) is : 90
Value of a + (b * c) / d is : 50

Parentheses for Grouping Subexpressions

Parentheses are used in C expressions in the same manner as in algebraic expressions. For example, to multiply a times the quantity b + c we write a * (b + c).

Rules of Operator Precedence

C applies the operators in arithmetic expressions in a precise sequence determined by the following rules of operator precedence, which are generally the same as those in algebra:

1. Operators in expressions contained within pairs of parentheses are evaluated first. Parentheses are said to be at the "highest level of precedence." In cases of nested, or embedded, parentheses, such as

   ```
   ( ( a + b ) + c )
   ```

   the operators in the *innermost* pair of parentheses are applied first.

2. Multiplication, division and remainder operations are applied next. If an expression contains several multiplication, division and remainder operations, evaluation proceeds from left to right. Multiplication, division and remainder are said to be on the same level of precedence.

3. Addition and subtraction operations are evaluated next. If an expression contains several addition and subtraction operations, evaluation proceeds from left to right. Addition and subtraction also have the same level of precedence, which is lower than the precedence of the multiplication, division and remainder operations.

4. The assignment operator (=) is evaluated last.

The rules of operator precedence specify the order C uses to evaluate expressions.[1] When we say evaluation proceeds from left to right, we're referring to the associativity of the operators. We'll see that some operators associate from right to left. Figure 2.10 summarizes these rules of operator precedence for the operators we've seen so far.

Math library functions

Math library functions allow you to perform certain common mathematical calculations.

Functions are normally used in a program by writing the name of the function followed by a left parenthesis followed by the argument (or a comma-separated list of arguments) of the function followed by a right parenthesis. For example, to calculate and print the square root of 900.0 you might write When this statement executes,

printf("%.2f", sqrt(900.0));

the math library function sqrt is called to calculate the square root of the number contained in the parentheses (900.0). The number 900.0 is the argument of the sqrt function. The preceding statement would print 30.00. The sqrt function takes an argument of type double and returns a result of type double. All functions in the math library that return floating-point values return the data type double. Note that double values, like float values, can be output using the %f conversion specification.

Error-Prevention Tip Include the math header by using the preprocessor directive #include when using functions in the math library

REFERENCE SITE: https://www.geeksforgeeks.org/c-library-math-h-functions/

Function	Description	Example
`sqrt( x )`	square root of x	`sqrt( 900.0 )` is 30.0 `sqrt( 9.0 )` is 3.0
`cbrt( x )`	cube root of x (C99 and C11 only)	`cbrt( 27.0 )` is 3.0 `cbrt( -8.0 )` is -2.0
`exp( x )`	exponential function e^x	`exp( 1.0 )` is 2.718282 `exp( 2.0 )` is 7.389056
`log( x )`	natural logarithm of x (base e)	`log( 2.718282 )` is 1.0 `log( 7.389056 )` is 2.0
`log10( x )`	logarithm of x (base 10)	`log10( 1.0 )` is 0.0 `log10( 10.0 )` is 1.0 `log10( 100.0 )` is 2.0
`fabs( x )`	absolute value of x as a floating-point number	`fabs( 13.5 )` is 13.5 `fabs( 0.0 )` is 0.0 `fabs( -13.5 )` is 13.5
`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 )` is 10.0 `ceil( -9.8 )` is -9.0
`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 )` is 9.0 `floor( -9.8 )` is -10.0
`pow( x, y )`	x raised to power y (x^y)	`pow( 2, 7 )` is 128.0 `pow( 9, .5 )` is 3.0
`fmod( x, y )`	remainder of x/y as a floating-point number	`fmod( 13.657, 2.333 )` is 1.992
`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 )` is 0.0
`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 )` is 1.0
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 )` is 0.0

Fig. 5.2 | Commonly used math library functions.