

Module 2

Operators

An Operator is a symbol that operates on a certain data type. The data items that operators act upon are called operands. Some operators require two operands, some operators act upon only one operand. In C, operators can be classified into various categories based on their utility and action.

- 1.Arithmetic Operators
- 2.Relational Operators3.Logical Operator
- 4.Assignment Operator
5. Increment & Decrement Operator
6. Conditional Operator
7. Bitwise Operator
8. Comma Operator

1.Arithmetic Operators

The Arithmetic operators performs arithmetic operations. The Arithmetic operators can operate on any built in data type. A list of arithmetic operators are

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo division

2.Relational Operators

Relational Operators are used to compare arithmetic, logical and character expressions. The Relational Operators compare their left hand side expression with their right hand side expression. Then evaluates to an integer. If the Expression is false it evaluate to "zero"(0) if the expression is true it evaluate to "one"

Operator	Meaning
<	Less than
>	Greater than
<=	Less than or Equal to
>=	Greater than or Equal to
=	Equal to
!=	Not Equal to

The Relational Operators are represented in the following manner:

Expression-1 Relational Operator Expression-2

The Expression-1 will be compared with Expression -2 and depending on the relation the result will be either "TRUE" OR "FALSE".

Examples :

Expression Evaluate to

(5 <= 10) ----- 1

(-35 > 10) ----- 0

(X < 10) ----- 1 (if value of x is less than 10) 0 Other wise

(a + b) == (c + d) 1 (if sum of a and b is equal to sum of c, d) 0 Other wise

3.Logical Operators

A logical operator is used to evaluate logical and relational expressions. The logical operators act upon operands that are themselves logical expressions. There are three logical operators.

Operators	Expression
&&	Logical AND
	Logical OR
!	Logical NOT

Logical And (&&): A compound Expression is true when two expression when two expressions are true. The && is used in the following manner.

Exp1 && Exp2.

The result of a logical AND operation will be true only if both operands are true.

The results of logical operators are:

Exp1	Op.	Exp2	Result
True	&&	True	True
True	&&	False	False
False	&&	False	False
False	&&	True	False

Example: a = 5; b = 10; c = 15;

Exp1	Op.	Exp2	Result
1. (a < b)	&&	(b < c) => True	
2. (a > b)	&&	(b < c) => False	
3. (a < b)	&&	(b > c) => False	
4. (a > b)	&&	(b > c) => False	

Logical OR(||): A compound expression is false when all expression are false otherwise the compound expression is true. The operator “||” is used as It evaluates to true if either exp-1 or exp-2 is true. The truth table of “OR” is Exp1 || Exp2

Exp1	Operator	Exp2	Result:
True		True	True
True		False	True
False		True	True
False		False	False

Example: a = 5; b = 10; c = 15;

Exp1	Op.	Exp2	Result
1. (a < b)		(b < c) => True	
2. (a > b)		(b < c) => True	
3. (a < b)		(b > c) => True	
4. (a > b)		(b > c) => False	

Logical NOT: The NOT (!) operator takes single expression and evaluates to true(1) if the expression is false (0) or it evaluates to false (0) if expression is true (1). The general form of the expression.

! (Relational Expression)

The truth table of NOT :

Operator.	Exp1	Result
!	True	False
!	False	True

Example: a = 5; b = 10; c = 15

- 1.!(a < b) => False
- 2.!(a > b) => True

4.Assignment Operator

An assignment operator is used to assign a value to a variable. The most commonly used assignment operator is =. The general format for assignment operator is :

<Identifier> = < expression >

Where identifier represent a variable and expression represents a constant, a variable or a Complex expression.

If the two operands in an assignment expression are of different data types, then the value of the expression on the right will automatically be converted to the type of the identifier on the left.

Example: Suppose that I is an Integer type Variable then

- 1.I = 3.3 3 => (Value of I)
- 2.I = 3.9 3 => (Value of I)
- 3.I = 5.74 5 => (Value of I)

Multiple assignment

< identifier-1 > = < identifier-2 > = - - - = < identifier-n > = <exp>;

Example: a,b,c are integers; j is float variable

1. `a = b = c = 3;`
2. `a = j = 5.6;` then `a = 5` and `j` value will be 5.6

C contains the following five additional assignment operators

1. `+=`
2. `-=`
3. `*=`
4. `/=`
5. `%=`

The assignment expression is: - `Exp1 < Operator> Exp-2`

Ex: `i = 10` (assume that)

Expression	Equivalent to	Final Value of 'i'
1. <code>i += 5</code>	<code>i = i + 5</code>	15
2. <code>i -= 5</code>	<code>i = i - 5</code>	10
3. <code>i *= 5</code>	<code>i = i * 5</code>	50
4. <code>i /= 5</code>	<code>i = i / 5</code>	10
5. <code>i %= 5</code>	<code>i = i % 5</code>	2

5.Increment & Decrement Operator

The increment/decrement operator act upon a Single operand and produce a new value is also called as "unary operator". The increment operator `++` adds 1 to the operand and the Decrement operator `--` subtracts 1 from the operand.

Syntax: `< operator >< variable name >;`

The `++` or `--` operator can be used in the two ways.

Example : `++ a`; Pre-increment (or) `a++` Post increment — `a`; Pre- Decrement (or) `a--` Post decrement

1. `++ a` Immediately increments the value of `a` by 1.
2. `a++` The value of the `a` will be increment by 1 after it is utilized.

Example 1: Suppose `a = 5` ;

Statements	Output
<code>printf ("a value is %d", a);</code>	a value is 5
<code>printf ("a value is %d", ++ a);</code>	a value is 6
<code>printf ("a value is %d ", a);</code>	a value is 6

Example 2: Suppose : `a = 5` ;

Statements	Output
<code>printf ("a value is %d ", a);</code>	a value is 5
<code>printf ("a value is %d ", a++);</code>	a value is 5
<code>printf ("a value is %d ",a);</code>	a value is 6

Similarly pre and post decrement work for an operand.

6.Conditional operator (or) Ternary operator (?:)

It is called ternary because it uses three expression. The ternary operator acts like If- Else construction.

Syntax : `<Exp -1> ? <Exp-2> : <Exp-3>);`

Expression-1 is evaluated first. If Exp-1 is true then Exp-2 is evaluated other wise Exp-3 will be evaluated.

Example:

1. `a = 5 ; b = 3;`
`(a > b ? printf ("a is larger") : printf ("b is larger"));`
 Output is :a is larger
2. `a = 3; b = 3;`
`(a > b ? printf ("a is larger") : printf ("b is larger"));`
 Output is :b is larger

7.Bit wise Operator

A bitwise operator operates on each bit of data. These bitwiseoperator can be divided into three categories.

i.The logical bitwise operators.

- ii. The shift operators
- iii. The one's complement operator.

i) The logical Bitwise Operator : There are three logical bitwise operators.

Meaning	Operator:
a) Bitwise AND	&
b) Bitwise OR	
c) Bitwise exclusive XOR	^

Suppose b1 and b2 represent the corresponding bits within the first and second operands, respectively.

B1	B2	B1 & B2	B1 B2	B1 ^ B2
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

The operations are carried out independently on each pair of corresponding bits within the operand thus the least significant bits (ie the right most bits) within the two operands. Will be compared until all the bits have been compared. The results of these comparisons are

A **Bitwise AND** expression will return a 1 if both bits have a value of 1.

Otherwise, it will return a value of 0.

A **Bitwise OR** expression will return a 1 if one or more of the bits have a value of 1. Otherwise, it will return a value of 0.

A **Bitwise EXCLUSIVE OR** expression will return a 1 if one of the bits has a value of 1 and the other has a value of 0. Otherwise, it will return a value of 0.

Example::

Variable	Value	Binary Pattern
X	5	0101
Y	2	0010
X & Y	0	0000
X Y	7	0111
X ^ Y	7	0111

ii) The Bitwise shift Operations: The two bitwise shift operators are **Shift left (<<)** and **Shift right (>>)**. Each operator requires two operands. The first operand that represents the bit pattern to be shifted. The second is an unsigned integer that indicates the number of displacements.

Example: `c = a << 3;`

The value in the integer a is shifted to the left by three bit position. The result is assigned to the c.

`A = 13; c = A << 3;`

Left shift `<< c = 13 * 23 = 104;`

Binary no 0000 0000 0000 1101

After left bit shift by 3 places ie, `a << 3` 0000 0000 0110 1000

The right –bit – shift operator (`>>`) is also a binary operator.

Example: `c = a >> 2 ;`

The value of a is shifted to the right by 2 position insert 0's Right – shift >> drop off 0's

0000 0000 0000 1101

After right shift by 2 places is `a >> 2` 0000 0000 0000 0011 `c = 13 >> 2` `c = 13/4 = 3`

iii) Bit wise complement: The complement op. `~` switches all the bits in a binary pattern, that is all the 0's becomes 1's and all the 1's becomes 0's.

variable	value	Binary pattern
x	23	0001 0111
~x	132	1110 1000

8. Comma Operator

A set of expressions separated by using commas is a valid construction in c language.

Example : `int i, j; i = (j = 3, j + 2) ;`

The first expression is $j = 3$ and second is $j + 2$. These expressions are evaluated from left to right. From the above example $l = 5$.

Size of operator: The operator size operator gives the size of the data type or variable in terms of bytes occupied in the memory. This operator allows a determination of the no of bytes allocated to various Data items

Example : `int i; float x; double d; char c;`

Statement	OUTPUT
<code>Printf ("integer : %d\n", sizeof(i));</code>	Integer : 2
<code>Printf ("float : %d\n", sizeof(i));</code>	Float : 4
<code>Printf ("double : %d\n", sizeof(i));</code>	double : 8
<code>Printf ("char : %d\n", sizeof(i));</code>	character : 1

Expressions

An expression can be defined as collection of data object and operators that can be evaluated to lead a single new data object. A data object is a constant, variable or another data object.

Example :

```
a + b
x + y + 6.0
3.14 * r * r
( a + b ) * ( a - b )
```

The above expressions are called as arithmetic expressions because the data objects (constants and variables) are connected using arithmetic operators.

Evaluation Procedure: The evaluation of arithmetic expressions is as per the hierarchy rules governed by the C compiler. The precedence or hierarchy rules for arithmetic expressions are

1. The expression is scanned from left to right.
2. While scanning the expression, the evaluation preference for the operators are
 - `*`, `/`, `%` - evaluated first
 - `+`, `-` - evaluated next

3. To overcome the above precedence rules, user has to make use of parenthesis. If parenthesis is used, the expression/ expressions with in parenthesis are evaluated first as per the above hierarchy.

Statements

Data Input & Output

An input/output function can be accessed from anywhere within a program simply by writing the function name followed by a list of arguments enclosed in parentheses. The arguments represent data items that are sent to the function.

Some input/output Functions do not require arguments though the empty parentheses must still appear. They are:

getchar()

Single characters can be entered into the computer using the C library Function `getchar()`. It returns a single character from a standard input device. The function does not require any arguments.

Syntax: `<Character variable> = getchar();`

Example:

```
char c;
c = getchar();
```

putchar()

Single characters can be displayed using function `putchar()`. It returns a single character to a standard output device. It must be expressed as an argument to the function.

Syntax: `putchar(<character variable>);`

Example:

```
char c;
-----
putchar(c);
```

gets()

The function gets() receives the string from the standard input device.

Syntax: gets(<string type variable or array of char>); Where s is a string.

The function gets accepts the string as a parameter from the keyboard, till a newline character is encountered. At end the function appends a “null” terminator and returns.

puts()

The function puts() outputs the string to the standard output device.

Syntax: puts(s);

Where s is a string that was read with gets();

Example:

```
main()
{
    char line[80]; gets(line); puts(line);
}
```

scanf()

Scanf() function can be used to input the data into the memory from the standard input device. This function can be used to enter any combination of numerical values, single characters and strings. The function returns number of data items.

Syntax: scanf (“control strings”, &arg1,&arg2,—&argn);

Where control string refers to a string containing certain required formatting information and arg1, arg2,—argn are arguments that represent the individual input data items.

Example: #include<stdio.h> main()

```
{
    char item[20];
    int partno;
    float cost;
    scanf(“%s %d %f”,&item,&partno,&cost);
}
```

Where s, d, f with % are conversion characters. The conversion characters indicate the type of the corresponding data. Commonly used conversion characters from data input.

Conversion Characters

Characters	Meaning
%c	data item is a single character.
%d	data item is a decimal integer.
%f	data item is a floating point value.
%e	data item is a floating point value.
%g	data item is a floating point value.
%h	data item is a short integer.
%s	data item is a string.
%x	data item is a hexadecimal integer.
%o	data item is a octal integer.

printf()

The printf() function is used to print the data from the computer’s memory onto a standard output device. This function can be used to output any combination of numerical values, single character and strings.

Syntax: printf(“control string”, arg-1, arg-2,—arg-n);

Where control string is a string that contains formatted information, and arg-1, arg-2 — are arguments that represent the output data items.

Example:

```
#include<stdio.h> main()
{
    char item[20]; int partno; float cost;
    _____
    printf(“%s %d %f”, item, partno, cost);
} (Where %s %d %f are conversion characters.)
```

Operator Precedence and Associativity

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

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.

OPERATOR	DESCRIPTION	ASSOCIATIVITY
() [] . -> ++ —	Parentheses (function call) (see Note 1) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement (see Note 2)	left-to-right
++ — + — ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (convert value to temporary value of <i>type</i>) Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ —	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
= += -= *= /= %= &= ^= = <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right