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

PROBLEM SOLVING AND C PROGRAMMING

I YEAR – I SEM

UNIT – II C PROGRAMMING BASICS

TOPIC – OPERATORS & EXPRESSIONS

OPERATORS

'C' supports a rich set of built-in operators. An operator is a symbol or a special character that tells the compiler 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.

The data items that operators act upon are called operands.

An expression is a combination of variables, constants and operators written according to the syntax of the language.

Types of Operators

In C, operators can be classified into various categories based on their utility and action. They are:

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

Integers, floating point numbers and double precision numbers can be added, subtracted, divided or multiplied. The operators used for these arithmetic operations are called arithmetic operators. C supports all basic arithmetic operators. The list of arithmetic operators and their meanings are given below:

Operator	**Meaning**
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulus Operators

Integer Arithmetic :

When both the operands in a single arithmetic expression such as a + b are declared as integers, the expression is called an integer expression. An arithmetic operation performed on these integer is called integer arithmetic and it always yields an integer value.

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 Program :

```
#include<stdio.h>

void main()

{

int a,b;

printf(“Enter any two numbers:”);

scanf(“%d%d”, &a, &b);

printf(“a+b=%d \n”,a+b);

printf(“a-b=%d \n”,a-b);
printf("a*b=%d \n",a*b);

printf("a/b=%d \n",a/b);

printf("a%b=%d \n",a%b);

}
```

Output :

```
Enter any two numbers: 10 5

a+b=15

a-b=5

a*b=50

a/b=2

a%b=0
```

2. Relational Operators

The relational operators are used to compare the value between two variables or between a variable and a constant. The list of relational operators and their meanings are given below:

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

An expression containing a relational operator is called as relational expression. The relational operators produce an integer result to express the condition of the comparison. If the condition is false, then the integer value is 0. If the condition is true, then the integer value is 1.

Syntax :

```
exp1 rel_op exp2
```

Where exp1 and exp2 are arithmetic expressions, which may be simple constants, variables or combination of both and rel_op represents a relational operator.

3. Logical Operators

Logical operators are used to combine two or more relations. The logical operators are called Boolean operators because the tests between values are reduced to either true or false, with zero being false and one being true. The list of logical operators and their meanings are given below:

Operator	**Meaning**
&&	Logical AND
||	Logical OR
!	Logical NOT

Example Program :

```
#include<stdio.h>
void main()
{
int a,b;
printf("\nEnter      the two  values:");
scanf("%d %d",&a,&b);
printf("\n (a>5)&&(b>2)    is%d",(a>5)&&(b>2));
printf("\n       (a>5)||(b>2)  is%d",(a>5)||(b>2));
printf("\n       !a is  %d",(!a));
}
```

Output :

```
Enter the two values: 7 4

(a>5)&&(b>2) is 1 (a>5)||(b>2) is 1

!a is 0
```

4. Assignment Operators

The assignment operator is used to assign the result of an expression to a variable. The most commonly used assignment operator is ‘=’.

Syntax:

var = exp ;

Where var is a variable and exp can be a constant or a variable or a complex expression.

Examples:

num = 25 ; age = 18 ; pi = 3.14 ; area = 3.14 * r * r ;

Example Program :

```
#include<stdio.h>

void main()

{

int a=10, b=20; int c,d,e;

c=a+b; // expression d=e=b; // nested assignment

a+=10; // compound assignment printf(“%d %d %d”, c,d,a);

}
```

Output :

```
30 20 20
```

5. Increment and Decrement Operators

C allows two useful unary operators generally not found in other computer languages. These are increment (++) and decrement (--) operators. The operator ++ adds 1 to its operand and -- subtracts 1 to its operand.

The increment and decrement operators and their meanings are given below:

Operator	**Meaning**
++	Increment operator (Adds 1)
--	Decrement operator (Subtracts 1)

Syntax :

(pre) ++variable_name;   (pre) --variable_name;

(or)

variable_name++(post);   variable_name--(post);

The operator is placed either before or after the variable name. If the operator placed before the variable like ++i or —i, it is known as pre- increment and the pre-decrement respectively. If the operator appears after the variable like i++ or i--, it is known as post increment and post decrement respectively.

This increment and decrement operator ++m and --m is equivalent to

++m is equivalent to m=m+1 or m+=1 --m is equivalent to m=m-1 or m-=1

This ++m and m++ means same thing when they form statement independently, they behave differently when they are used in expression.

Consider m = 6 y = ++m,

In this case the value of y is 7 and m is also 7. The pre-increment operator first adds to the operand and then the result is assigned to the variable on left.

Consider m=5, y=m++

then the value of y=5, and value of m=6. The post increment operator first assigns the value to the variable on te left and the increments the operand.

Example Program :

```
#include<stdio.h>

void main()

{

int a,b,c;

printf("\n enter the two values:");

scanf("%d%d",&a,&b);

c=(a>b)?a:b;

printf("\n(a>b)?a:b is %d",c);

}
```

Output:

```
Enter the two values:

3

4

(a>b)?a:b is 4.
```

6. Conditional Operators

The conditional operator is also known as a **ternary operator**. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., '?' and ':'.

As conditional operator works on three operands, so it is also known as the ternary operator.

The behavior of the conditional operator is similar to the 'if-else' statement as 'if-else' statement is also a decision-making statement.

Syntax of a conditional operator

1. Expression1? expression2: expression3;

In the above syntax, the expression1 is a Boolean condition that can be either true or false value.
If the expression1 results into a true value, then the expression2 will execute.
The expression2 is said to be true only when it returns a non-zero value.
If the expression1 returns false value then the expression3 will execute.
The expression3 is said to be false only when it returns zero value.

Example

```
#include <stdio.h>

int main()

{

   int age;  // variable declaration

   printf("Enter your age");

   scanf("%d",&age);   // taking user input for age variable
   (age>=18)? (printf("eligible for voting")) : (printf("not eligible for voting"));  // conditional
 operator

   return 0;

}
```

7. Bitwise Operators

One of C’s powerful features is a set of bit manipulation operators. This permits the programmer to access and manipulate individual bits within a piece of data. The bitwise operators can operate upon int and char data types but not on float and double data types. The list of bitwise operators and their meanings are given below:

Operator	**Meaning**
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	One’s complement

All operators except ~ operator are binary operators which requires two operands. While using the bit operators, each operand is treated as a binary number consisting of a series of individual 1s and 0s. The respective bits in each operand are then compared on a bit by bit basis and result is determined based on the selected operation.

Example Program 1:

Write a program to shift the input data two bits right.

```
#include<stdio.h>

#include<conio.h>

void main()

{

int x,y; clrscr();

printf(“Read the integer”);
scanf("%d",&x);

x>>2;

y=x;

printf("Right shifted data is %d",y);

}
```

Output :

```
Read the integer 8

Right shifted data is 2
```

Example Program 2:

Write a program to use bitwise AND.

```
#include<stdio.h>

#include<conio.h>

main()

{

int a,b,c; clrscr();

printf("Read the values i"); scanf("%d %d", &a,&b); c=a&b;

printf("The values after ANDing is c=%d",c);

}
```

Output :

```
Read the value : 8 4

The values of the ANDing is c = 0
```

8. Special Operators

C supports some special operators such as comma operators, size of operator, pointer operators (& and *) and member selection operators (. and) The comma and size of operator is discussed below:

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.

Example :

```
sum = (a = 10, b = 20, a + b) ;
```

Here, the statement first assigns the value 10 to a, then assigns 20 to b, and finally assigns 30 (i.e. 10 + 20) to sum. Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

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. The syntax is :

```
sizeof(expression) ; (or)sizeof(data_type) ;
```

Example :

```
a= sizeof(sum) ;

b= sizeof(long int) ;

c= sizeof(235L) ;
```

The sizeof operator is normally used to determine the length of arrays and structures, when their sizes are not known in advance by the programmer. It is also used to allocate memory space dynamically to variables during execution of a program.

Example Program :

```
#include<stdio.h>

void main()

{

int a;

printf("\nEnter a value"); scanf("%d",&a);
printf("\nThe sizeof(a) is %d",sizeof(a));

}
```

Output :

```
Enter a value 1

The sizeof(a) is 2
```

The Difference between Operands and Operators

Operands	**Operators**
Operands are the data values.	Operator is a symbol or sign that mean by any operation.
Operands are the variables or constants on which the operation is performed.	Operands are the variables or values between which the operation is performed.
Example: sum=a+b;c The variables a and b are the operands	Example: sum=a+b;c The arithmetic sign+ is the arithmetic addition operator.

The Difference between Signed and Unsigned Integer

Signed Integer	**Unsigned Integer**
Signed integer is a whole number whose value lies in the range from negative to a positive integer.	Unsigned integer is a whole number that is greater than or equal to zero.
It carries a positive or negative sign.	It does not carry a positive or negative sign,
It can represent both negative and positive numbers.	It can carry only zero and positive numbers.
It ranges from-32,768 to +32,767.	It ranges form 0 to 65,535.
% d prints as signed integer	% u print an unsigned integer

TYPE CONVERSIONS IN EXPRESSIONS

a) Implicit Type Conversion

C permits mixing of constants and variables of different types in an expression. C automatically converts any intermediate values to the proper type so that the expression can be evaluated without loosing any significance. This automatic type conversion is known as implicit type conversion.

The following Rules apply during Evaluating Expressions.

All short and char are automatically converted to int then

1. If one operand is long double, the other will be converted to long double and result will be long double.

2. If one operand is double, the other will be converted to double and result will be double.

3. If one operand is float, the other will be converted to float and result will be float.

4. If one of the operand is unsigned long int, the other will be converted into unsigned long int and result will be unsigned long int.

5. If one operand is long int and other is unsigned int then

(a)If unsigned int can be converted to long int, then unsigned int operand will be converted as such and the result will be long int.

(b)Else both operands will be converted to unsigned long int and the result will be unsigned long int.

6. If one of the operand is long int, the other will be converted to long int and the result will be long int.

7. If one operand is unsigned int the other will be converted to unsigned int and the result will be unsigned int.

b) Explicit Conversion

Many times there may arise a situation where we want to force a type conversion in a way that is different from automatic conversion.

Syntax:

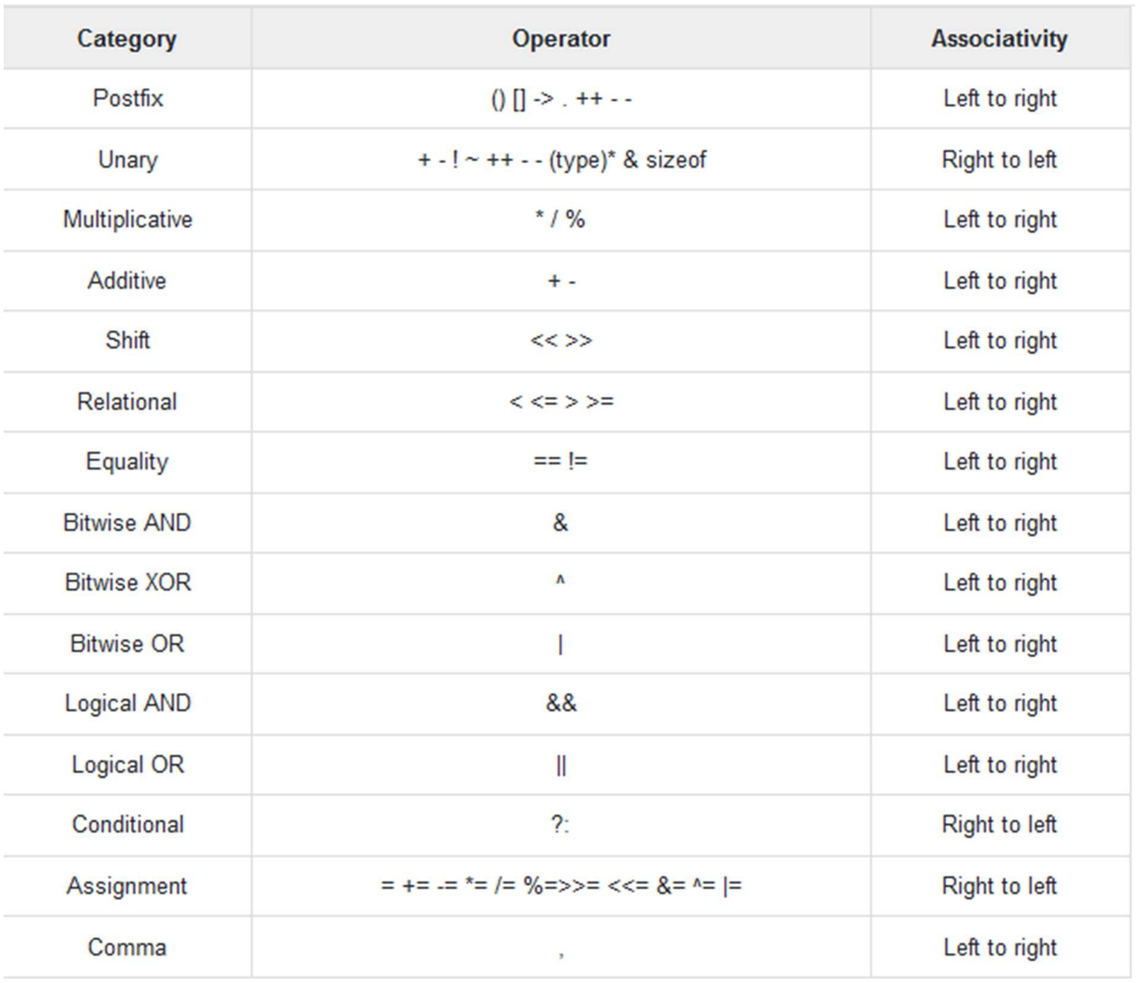
(type_name) expression

PRECEDENCE OF OPERATORS

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