4

TYPES OF OPERATORS

Unit Structure

4.0 OBJECTIVES

This chapter would make you understand the following concepts:

- To design programs to perform basic calculations.
- To compare two entity and implement decision making order of evaluation of operators.

4.1 TYPES OF OPERATORS

C Programming language supports a set of built-in operators. An operator is a symbol that operates on the operands and instructs the compiler to perform certain operations. Operands can be any variables that hold value.

The various types of Operators are as follows:

1. Arithmetic Operator
2. Relational Operator
3. Logical Operator
4. Compound Assignment Operator
5. Increment/Decrement Operator
6. Conditional & ternary Operator
7. Bitwise & comma operator

4.1.1 Arithmetic Operator:

This operator supports performing basic mathematical operations. Following are the operators used for arithmetic operations:

Symbol	Description	Example
Assume a=10 and b=5		
+	Addition of two operand values	a+b Output: 15
-	Subtraction of two operand values	a-b Output: 5
*	Multiplication of two operand values	a*b Output: 50
/	Division (Quotient) of two operand values	a/b Output: 2
%	Modulus (Remainder) of two operand values	a%b Output: 0 (Since a is divisible by b)

Example:

Program to calculate Simple Interest

Needed for getch();

```
#include<stdio.h>
#include <conio.h>
void main()
{
int p, n;
float r, Simple_interest;
p=10000;
n=10;
r=5.5;
Simple_interest = (p* n * r)/100;
printf("Simple Interest = %f",Simple_interest);
getch();
}
```

4.1.2 Relational Operator:

Relational operator is also known as Comparison operator. It performs comparison between two operand values:

Operator	Description	Example
Assume a=10 and b=5		
==	Check if two operand values are equal	a==b Output: false
!=	Checks if two operands values are not equal	a!=b Output: true
>	Checks if left operand value is greater than right operand value	a>b Output: true
<	Checks if left operand value is smaller than right operand value	a<b Output: false
>=	checks if left operand value is greater than or equal to right operand value	a>=b Output: true
<=	Check if left operand value is smaller than or equal to right operand value	a<=b Output: false

Example:

```
#include<stdio.h>
#include <conio.h>
void main()
{
int a, b;
a=10;
b=20;
printf("Value = %d", a<=b);
printf("Value = %d", a==b);
printf("Value = %d", a>b);
printf("Value = %d", a!=b);
getch();
}
```

Output:

Value = 1

Value = 0

Value = 0

Value = 1

4.1.3 Compound Assignment Operator:

Assignment operators in C language are as follows:

Operator	Description	Example
=	assigns values from right side to left side operand	a=10
Shorthand Assignment Operators		
+=	adds right operand value to the left operand value and assign the result to left operand	a+=b is same as a=a+b
-=	subtract right operand value to the left operand value and assign the result to left operand	a-=b is same as a=a-b
*=	multiply right operand value to the left operand value and assign the result to left operand	a*=b is same as a=a*b
/=	divide right operand value to the left operand value and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus right operand value to the left operand value and assign the result to left operand	a%=b is same as a=a%b

a = 10
b = 5
a += b
↳ a becomes 15.
Because,
a += b is same as
a = a + b

Example:

```
#include<stdio.h>
void main()
{
int n;
n=100;
n+=10;
printf("Value of n = %d",n);
getch();
}
```

Output:

Value of n = 110

4.1.4 Conditional & ternary Operator:

The conditional operator is also known as Ternary Operator or ?: operator
It is similar to the C language decision making, the if condition.

Syntax:

condition ? true statement : false statement

Explanation:

- condition is specified in the if followed by question mark "?". Based on the condition, the statements are executed.
- True statements are executed (statement after question mark "?") only if the condition returns true as the boolean value.
- False statements are executed (statement after question mark ":") only if the condition returns false as the boolean value.

con ? iftrue : iffalse

con → condition which evaluates to true or false.

iftrue → executes if con is true

iffalse → executes if con is false

Example:

```
#include<stdio.h>

void main()

{

int n=30;

if(n%3==0) ? printf("Number is divisible by 3") : printf("Number is not divisible by 3");

getch();

}
```

Output:

Number is divisible by 3

4.1.5 Increment/Decrement Operator:

C programming supports increment operator ++ and decrement operator --

These two operators operate on a single operand only

- Increment ++ increases the value of the operand by 1
- Decrement -- decreases the value of operand by 1

a = 10
a++
↑ a becomes 11

b = 20
b--
↳ b becomes 19.

Example:

```
#include<stdio.h>

int main()

{
```

```
int a,b;
a=10;
b=20;
printf("Value of a = %d", a++);
printf("Value of b = %d", b--);
return 0;
}
```

Output:

Value of a = 11

Value of b = 19

4.1.6 Logical Operator:

Operator	Description	Example
Assume a=1 and b=0		
&&	Logical AND If both the conditions are true then it returns true. If either of the condition is false then it returns false	(A && B) is false
||	Logical OR If both or either of the conditions are true then it returns true. If both the conditions are false then it returns false	(A || B) is true.
!	Logical NOT It gives the negation of an expression ie. if an expression is true then it returns false and vice versa.	!(A && B) is true.

only true when both true

only false when both false

negation → changes true to false and vice versa.

Example:

```
#include<stdio.h>
void main()
{
int n;
n=35;
```

```
if (n%5==0 && n%7==0) ? printf("Number is divisible by both") :
printf("Number is divisible by none");

}
```

Output:

Number is divisible by both

4.1.7 Bitwise & comma operator:

Bitwise operator operates on bits and performs bit by bit operation:

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

a	b	a&b	a|b	a^b
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

4.2 PRECEDENCE & ORDER OF EVALUATION

- Operator precedence defines the order of an expression to be evaluated.
- Some operators have higher precedence than other operators. Example, the addition operator has a higher precedence than the subtraction operator.
- The operator with higher precedence is evaluated first and the operator with lower precedence is evaluated.
- For example, x = 10 + 5 * 2; here, x is assigned value as 20 and not 30 because multiplication operator (*) has a higher precedence than addition operator (+)

- Hence, first 5*2 is calculated and then it's added to 10.
- In the table, the operators with the highest precedence appear on the top of the table and those with the lowest precedence appear at the bottom. First the higher precedence operators will be evaluated and then the lower precedence operator.

In the table, associativity determines the precedence value of the list of operators under the categories:

Type	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

4.3 STATEMENT & EXPRESSIONS

Expressions:

In any programming language, an 'expression' is basically a combination of variables and operators that are interpreted by the compiler.

For example:

expression alone does not do anything significant.

c=a+b

a<=b

a++

Here, a and b are the variables having some values and +, <=, ++ are the operators which manipulate the values of the variable. And the complete unit is called an expression.

Statements:

A 'statement' is a standalone unit for execution.

For Examples

A statement can be the jump statements; return, break, continue, and goto.

a=10

4.4 AUTOMATIC & EXPLICIT TYPE CONVERSION

Type conversion means converting one operand of a data type into another one. It is also called type casting or type conversion in C language

C programming provides two types of type conversion:

1. Implicit type (Automatic) casting
2. Explicit type casting

1. Implicit type (Automatic) casting:

In Implicit type conversion, the operand of one data type is converted into another data type automatically but an operand of lower data type is converted into a higher data type automatically.

Example:

```
#include<stdio.h>
int main(){
    //initializing variable of short data type
    short a=10;
    int b; //declaring int variable
```

```
    b=a; //implicit type casting
    printf("Value of a = %d\n",a);
    printf("Value of b = %d\n",b);
    return 0;
}
```

Output:

Value of a = 10

Value of b = 10

2. Explicit type casting:

In Explicit type casting, one datatype is forcefully type casted to another datatype.

Example:

```
#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

4.5 MISCELLANEOUS PROGRAMS

Practice Program:

1. Write a program to check whether a number is even or odd

```
#include<stdio.h>
void main()
{
int n;
```

```
n=35;
if (n%2==0) ? printf("Number is even") : printf("Number is odd");
}
```

Output:

Number is even

2. Write a program to check whether the alphabet is a vowel or not

```
#include<stdio.h>
void main()
{
char ch;
ch='a';
if (ch=='a' || ch=='e'|| ch=='i'|| ch=='o'||ch=='u' ) ? printf("It is a vowel")
: printf("It is not a vowel");
}
```

Output:

It is a vowel

4.6 SUMMARY

Operators are defined as symbols that operate on operands which help us to perform specific mathematical and logical computations.

C language works with majorly 7 different types of operators namely Arithmetic, Relational, Logical, Assignment, Increment/Decrement, Conditional and Bitwise operator.

The precedence of operators matters in the grouping and evaluation of expressions.

First the expressions of higher-precedence operators are evaluated and if there are several operators having equal precedence, then they are evaluated from left to right.

Typecasting is converting one data type into another one.

C language uses two types of typecasting namely implicit and explicit conversion.