Unit 4

Operators and Expression

C Operators

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.
- These C operators join individual constants and variables to form expressions.
- An operator is a symbol that instructs C to perform some operation, or action, on one or more operands.
- An operand is something that an operator acts on.
- Consider the expression A + B * 5. where, +, * are operators, A, B are variables, 5 is constant and A + B * 5 is an expression

TYPES OF C OPERATORS

- Arithmetic operators
- Assignment operators
- Unary Operators
- Relational operators
- Logical operators
- Bit wise operators
- Conditional operators (ternary operators)
- Special operators

ARITHMETIC OPERATORS IN C

- C Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in C programs.

Operator	**Meaning of Operator**
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulos division)

Example

```
#include<stdio.h>
 void main()
{
   int a=40,b=20, add,sub,mul,div,mod;
   add = a+b;
   sub = a-b;
   mul = a*b;
   div = a/b;
   mod = a%b;
   printf("Addition of a, b is : %d\n", add);
   printf("Subtraction of a, b is : %d\n", sub);
   printf("Multiplication of a, b is : %d\n", mul);
   printf("Division of a, b is : %d\n", div);
   printf("Modulus of a, b is : %d\n", mod);
}
```

Example 2

```
#include<stdio.h>
void main()
{
   int result;
   result = 6%4;
   printf("6%%4 = %d\n",result);
   result = 4%6;
   printf("4%%6 = %d\n",result);
   result = 6%3;
   printf("6%%3 = %d\n",result);
   result = 3%6;
   printf("3%%6 = %d\n",result);
   result = 1%3;
   printf("1%%3 = %d\n",result);
   result = 3%1;
```

ASSIGNMENT OPERATORS IN C

- In C programs, values for the variables are assigned using assignment operators.
- For example, if the value “10” is to be assigned for the variable “sum”, it can be assigned as “sum = 10;”
- There are 2 categories of assignment operators in C language. They are,
  1. Simple assignment operator (Example: =)
  2. Compound assignment operators (Example: +=, -=, *=, /=, %=, &=, ^=)

Operator	Example	Same as
=	a = b	a = b
+=	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

```
#include <stdio.h>
void main()
{
    int a = 5, c;

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

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

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

    c *= a; // c = c*a
```

C Unary Operators

- The operators acting upon a single operand are called unary operators.
- The unary plus(+), unary minus(-),increment(++),decrement(--),sizeof, and address(&) operators are the common unary operators.

- **Increment and Decrement Operators**
- There are two special unary operators in C, increment and decrement which cause the variable they act on to be incremented or decremented by 1 respectively.
- x++; //equivalent to x=x+1;
- x--; //equivalent to x=x-1;

- ++ and -- can be used in prefix and postfix notation
- In prefix notation the value of the variable is either incremented or decremented and is then read.
- In prefix the operator is written before its operand (++x)/(--x).
- While in postfix notation the value of the variable is read first and is then incremented or decremented.
- In postfix the operator is written after its operand (x++)/(x--).

Expression	Operation	Interpretation

j=++k	Preincrement	k=k+1;
		j=k;

j=k++	Postincrement	j=k;
		k=k+1;

j=--k	Predecrement	k=k-1;
		j=k;

j=k--	Postdecrement	j=k;
		k=k-1;

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 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

Example

```
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, z;

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

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

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



    z=a < c;
```

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 of Operator	Example
&&	Logial 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

```
#include <stdio.h>
void main()
{
    int a = 5, b = 5, c = 10, result;
    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) equals to %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) equals to %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) equals to %d \n", result);

    result = !(a != b);
    printf("!(a != b) equals to %d \n", result);

    result = !(a == b);
    printf("!(a == b) equals to %d \n", result);
}
```

Bitwise Operators

- During computation, mathematical operations like: addition, subtraction, addition and division 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

Bitwise AND operator &

- The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.
- Let us suppose the bitwise AND operation of two integers 12 and 25.

```
 12 = 00001100 (In Binary)
 25 = 00011001 (In Binary)
 Bit Operation of 12 and 25
   00001100
 & 00011001
 ________
   00001000 = 8 (In decimal)
```

```c
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a&b);
    return 0;
}
```

Bitwise OR operator |

- The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

```
12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

   00001100

 | 00011001
     ________________________

   00011101 = 29 (In decimal)
```

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a|b);
    return 0;
}
```

Bitwise XOR (exclusive OR) operator ^

- The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25

00001100

^ 00011001

____

00010101 = 21 (In decimal)

Bitwise complement operator ~

- Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

35 = 00100011 (In Binary)

Bitwise complement Operation of 35

~ 00100011

____

11011100 = 220 (In decimal)

- **Twist in bitwise complement operator in C Programming**
- The bitwise complement of 35 (~35) is -36 instead of 220, but why?
- For any integer n, bitwise complement of n will be -(n+1). To understand this, you should have the knowledge of 2's complement.
- **2's Complement**
- Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1.

Shift Operators in C programming

- There are two shift operators in C programming:
- Right shift operator
- Left shift operator.
- **Right Shift Operator**
- Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

  212 = 11010100 (In binary)

  212>>2 = 00110101 (In binary) [Right shift by two bits]

  212>>7 = 00000001 (In binary)

  212>>8 = 00000000

  212>>0 = 11010100 (No Shift)

- **Left Shift Operator**
- Left shift operator shifts all bits towards left by certain number of specified bits. It is denoted by <<.

  212 = 11010100 (In binary)

  212<<1 = 110101000 (In binary) [Left shift by one bit]

  212<<0 =11010100 (Shift by 0)

  212<<4 = 110101000000 (In binary) =3392(In decimal)

Example

```
#include <stdio.h>
void main()
{
   int num=212, i;
   printf("Right shift by %d: %d\n", 0, num>>0);
   printf("Right shift by %d: %d\n", 1, num>>1);
   printf("Right shift by %d: %d\n", 2, num>>2);

   printf("\n");

   printf("Left shift by %d: %d\n", 0, num<<0);
   printf("Left shift by %d: %d\n", 1, num<<1);
   printf("Left shift by %d: %d\n", 2, num<<2);
}
```

Other Operators

- **Comma Operator**
- Comma operators are used to link related expressions together. For example:
- int a, c = 5, d;

- **The sizeof operator**
- The sizeof is an unary operator which returns the size of data (constant, variables, array, structure etc).

C Ternary Operator (?:)

- Ternary operator is a conditional operator that works on 3 operands.
- **Conditional Operator Syntax**
- conditionalExpression ? expression1 : expression2

The conditional operator works as follows:

- The first expression conditionalExpression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.
- If conditionalExpression is true, expression1 is evaluated.
- If conditionalExpression is false, expression2 is evaluated.

Example

```
#include<stdio.h>
void main()
{
    int n;
    printf("enter a number");
    scanf("%d",&n);
    n%2==0?printf("%d is even",n):printf("%d is odd",n);
}
```

Operator Precedence and Associativity

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

Operator	Meaning of operator	Associativity
() [] -> .	Functional call , Array element reference , Indirect member selection , Direct member selection	Left to right
! ~ + - ++ -- & * sizeof (type)	Logical negation, Bitwise(1 's) complement, Unary plus, Unary minus, Increment, Decrement, Dereference Operator(Address), Pointer reference, Returns the size of an object, Type cast(conversion)	Right to left
* / %	Multiply, Divide, Remainder	Left to right
+ -	Binary plus(Addition), Binary minus(subtraction)	Left to right
<< >>	Left shift, Right shift	Left to right
< <= > >=	Less than, Less than or equal, Greater than, Greater than or equal	Left to right
== !=	Equal to, Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND	Left to right
||	Logical OR	Left to right
?:	Conditional Operator	Right to left
= *= /= %= -= &= ^= |= <<= >>=	Simple assignment, Assign product, Assign quotient, Assign remainder, Assign sum, Assign difference, Assign bitwise AND, Assign bitwise XOR, Assign bitwise OR, Assign left shift, Assign right shift	Right to left

TypeCasting in C

- Typecasting is converting one data type into another one. It is also called as data conversion or type conversion in C language. It is one of the important concepts introduced in ‘C’ programming.
- ‘C’ programming provides two types of type casting operations:
- Implicit type casting
- Explicit type casting

Implicit type casting

- Implicit type casting means conversion of data types without losing its original meaning.
- This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.
- Implicit type conversion in C happens automatically when a value is copied to its compatible data type.
- During conversion, strict rules for type conversion are applied.
- If the operands are of two different data types, then an operand having lower data type is automatically converted into a higher data type.

Example

```
#include<stdio.h>
int main()
{
    short a=10;
    //initializing variable of short data type
    int b;
    //declaring int variable
    b=a; //implicit type casting
    printf("%d\n",a);
    printf("%d\n",b);
}
```

Explicit type casting

- In implicit type conversion, the data type is converted automatically. There are some scenarios in which we may have to force type conversion.
- Suppose we have a variable div that stores the division of two operands which are declared as an int data type.
- int result, var1=10, var2=3;
- result=var1/var2;
- In this case, after the division performed on variables var1 and var2 the result stored in the variable “result” will be in an integer format. Whenever this happens, the value stored in the variable “result” loses its meaning because it does not consider the fraction part which is normally obtained in the division of two numbers.
- To force the type conversion in such situations, we use explicit type casting.
- It requires a type casting operator. The general syntax for type casting operations is as follows:
- (type-name) expression