3. Operators and Expressions

- Introduction to operators and expressions
- Arithmetic, relational and logical operators
- Assignment, increment and decrement operators
- Conditional, bitwise and special operators
- Comma operator, size of operator
- Evaluation and type conversion in expressions
- Operator precedence and associativity

3.1 Introduction to operators and expressions

Operator:

An operator is a symbol that performance a specific operation on one or more operands.

Operand:

An operand is a variable or constant on which an operator works.

Expression:

An expression is a combination of operators and operands that produces a value.

Examples:

```
int result = a + b;
```

3.1 Introduction to operators and expressions

On the basis of number of operand :

- Unary operators
- Binary operators
- Ternary operators

Unary operators:

eg : ++(increment), --(decrement)

Binary operators:

eg : +,-,*,/

Ternary operator:

syntax:

`(condition ? expr1 : expr2)` //conditional operator

code:

```c
#include<stdio.h>

int main(){
    int a = 10;

    printf("A = %d,",a);
    printf("A = %d,",a++);
    printf("A = %d,",++a);
    printf("A = %d,",a--);
    printf("A = %d,\n",--a);

    int b = 10;
    int sum = a+b;
    int sub = a-b;

    b =(b>10?printf("True"):printf("False"));
    b =(b>10?sum:sub);

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

    return 0;
}
```

Q1:

Find the largest among two numbers using ternary operator?

Q2:

Find the largest among three numbers using ternary operator?

3.2 Arithmetic, relational and logical operators

On the basis of function, utility and actions:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and Decrement operators
6. Conditional operator
7. Bitwise Operator
8. Special Operator

Arithmetic operator: Used to perform mathematical calculations.

+(addition or unary plus)
-(subtraction or unary minus)
*(multiplication)
/(division)
%(modulo division, remainder) → % works only with integer data types

3.2 Arithmetic, relational and logical operators

Relational Operators: Used to compare two values. The result is true (1) or false (0).

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

Logical Operators: Used to combine multiple conditions.

&&	Logical AND
!	Logical NOT

3.2 Arithmetic, relational and logical operators

Truth Table: For AND(&&)

Truth Table: For OR(||)

Truth Table: For NOT(!)

3.3 Assignment, Increment, and Decrement Operators

Assignment Operators: Assignment operators are used to assign values to variables.

=	assignment operator	
compund		
+=	a += 5	a = a + 5
-=	a -= 3	a = a - 3
*=	a *= 2	a = a * 2
/=	a /= 2	a = a / 2
%=	a %= 2	a = a % 2

Increment Operator (++): Used to increase a variable's value by 1.

Types:

- Pre-increment: ++a
- Post-increment: a++

Decrement Operator (--): Used to decrease a variable's value by 1.

Types:

- Pre-decrement: --a
- Post-decrement: a--

3.4 Conditional, Bitwise, and Special Operators

Conditional Operator (?:): The conditional operator is also called the ternary operator.It is a short form of if–else.

Syntax:

```
→ condition ? expression1 : expression2;
```

Working

- If condition is true, expression1 is executed
- If condition is false, expression2 is executed

Bitwise Operators: Bitwise operators work on binary representation of numbers.x

&	Bitwise AND
^	Bitwise XOR
~	Bitwise NOT
<<	Left shift
>>	Right shift

```
int a = 5, b = 3;
printf("%d", a & b);
```

3.4 Conditional, Bitwise, and Special Operators

Special Operators:

sizeof Operator: Used to find the size of a data type or variable (in bytes).

```
printf("%d", sizeof(int));
```

Comma Operator: Allows multiple expressions to be evaluated in a single statement.

```
int a, b;
a = (b = 3, b + 2);
```

Address Operator (&): Used to find the memory address of a variable.

```
int x = 10;
printf("%u", &x);
```

Dereference Operator (*): Used with pointers to access the value at an address.

```
int x = 10;
int *p = &x;
printf("%d", *p);
```

3.5 Comma Operator and sizeof Operator

Comma Operator (,)

The comma operator allows multiple expressions to be evaluated in a single statement.
The value of the last expression is assigned or returned.

Example:

```
int a, b;
a = (b = 5, b + 2);

printf("%d", a);
```

Common Use In for loop:

```
for (i = 0, j = 10; i < j; i++, j--) {
    printf("%d %d\n", i, j);
}
```

3.5 Comma Operator and sizeof Operator

sizeof Operator

The sizeof operator is used to find the memory size (in bytes) of a data type or variable.

Example:

```
printf("%d", sizeof(int));
printf("%d", sizeof(float));
```

Example with Variable:

```
int x;
printf("%d", sizeof(x));
```

3.6 Evaluation and Type Conversion in Expressions

Expression Evaluation

Evaluation means how an expression is calculated to produce a result.
The compiler evaluates expressions based on:

- Data types
- Type conversion
- Operator precedence (covered in next section)

Type Conversion in Expressions
When an expression contains different data types, C automatically converts them to a common type before evaluation.

Types:

1. Implicit Type Conversion
2. Explicit Type Conversion

3.7 Operator Precedence and Associativity

Operator Precedence: Operator precedence decides which operator is evaluated first when multiple operators appear in an expression.

Example:

```
int result = 10 + 5 * 2;
```

Operator Associativity: Operator associativity decides the order of evaluation when two operators of the same precedence appear in an expression.

Example:

```
int result = 100 / 5 % 2;
```

Both in one expression:

Example:

```
int result = 100 + 200 / 10 - 3 * 10;
```

Associativity is only used when there are two or more operators of the same precedence.

3.7 Operator Precedence and Associativity

Operator	Description	Precedence level	Associativity
( ) [ ] . -> ++ --	Parentheses: grouping or function call Brackets (array subscript) Dot operator (Member selection via object name) Arrow operator(Member selection via pointer) Postfix increment/decrement	1	Left to Right
+ - ++ -- ! ~ * & (datatype) sizeof	Unary plus Unary minus Prefix increment/decrement Logical NOT One's complement Indirection Address (of operand) Type cast Determine size in bytes on this implementation	2	Right to Left
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right
<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right

3.7 Operator Precedence and Associativity

== !=	Equal to Not equal to	7	Left to Right
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
?:	Conditional operator	13	Right to Left
= *= /= %= += -= &= ^= |= <<= >>=	Assignment operators	14	Right to Left
,	Comma operator	15	Left to Right

Code:

```
#include<stdio.h>

int main() {

int res;
res = 2 + 4 > 3 && 5 == 5;
printf("Result=%d\n", res);
return 0;

}
```

Q1. find out final value of a, b and c by running sequentially.

```
int a = 2, b= 3, c;
a = (b++) + (++b) +a;
c = a>b ?a:b;
b = (a++)+(b--)+a;
c = c++*b--;
```