Unit 4
Operators and Expressions

Operators

- An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.

Arithmetic operator

- Arithmetic operators are used to perform arithmetic/mathematical operations on operands.

Operator	**Description**
+	Adds two operands.
−	Subtracts second operand from the first.
*	Multiplies both operands.
/	Divides numerator by de-numerator.
%	Modulus Operator and remainder of after an integer division.
++	Increment operator increases the integer value by one.
--	Decrement operator decreases the integer value by one.

Example:

```
/* Program to demonstrate the arithmetic operator */
#include <stdio.h>
int main()
{
        int a = 21;
        int b = 10;
        int c ;
        c = a + b;
        printf("Value of a + b is: %d\n", c );
        c = a - b;
        printf("Value of a - b is: %d\n", c );
        c = a * b;
        printf("Value of a * b is: %d\n", c );
        c = a / b;
        printf("Value of a / b is: %d\n", c );
        c = a % b;
        printf("Value of a %% b is: %d\n", c );
        c = a++;
        printf("Value of a++ is: %d\n", c );
        c = a--;
        printf("Value of a-- is: %d\n", c );
}
```

Output:

```
        Value of a + b is: 31
        Value of a - b is: 11
        Value of a * b is: 210
        Value of a / b is: 2
        Value of a % b is: 1
        Value of a++ is: 21
        Value of a-- is: 22
```

Relational operator

- Relational Operators are the operators used to create a relationship and compare the values of two operands.
- Following are the various types of relational operators in C.

Operator	Description
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.

Example:

```
/* Program to demonstrate the concept of relational operator */

#include <stdio.h>
int main() {
        int a = 21;
        int b = 10;
        int c ;
        if(a == b) {
        printf("a is equal to b\n" );
        }else{
        printf("a is not equal to b\n" );
        }
        if(a < b) {
        printf("a is less than b\n" );
        } else {
        printf("a is not less than b\n" );
        }
        if(a > b) {
        printf("a is greater than b\n" );
        } else {
        printf("a is not greater than b\n" );
        }
        if (a <= b) {
        printf("a is either less than or equal to  b\n" );
        }
        if (a >= b) {
        printf("b is either greater than  or equal to b\n" );
        }
}
```

__Output:__

```
        a is not equal to b
        a is not less than b
        a is greater than b
        b is either greater than  or equal to b
```

Logical or Boolean operator

- Boolean operators AND, OR, and NOT are used to manipulate logical statements.
- Boolean operators are the core operators used in digital control systems as well as computer systems.
- AND and OR are binary operators, while NOT is a unary operator.

Operator	**Description**
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

__Example:__

```
#include <stdio.h>
int main() {
        int a = 4;
        int b = 8;
        int c ;
        if (a && b) {
              printf("Condition is true\n");
        }
        if(a || b) {
              printf("Condition is true\n");
        }
        if (!(a && b)) {
          printf("Condition is true\n" );
        }
}
```

Output:

```
      Condition is true
      Condition is true
```

Assignment Operator

- The assignment operator is used to assign the value, variable and function to another variable.

Operator	Description
=	Simple assignment operator. Assigns values from right side operands to left side operand.
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.

Example:

```
#include <stdio.h>
int main() {
      int a = 21;
      int c ;
      c =  a;
      printf("= Operator Example, Value of c = %d\n", c );
      c +=  a;
      printf("+= Operator Example, Value of c = %d\n", c );
      c -=  a;
      printf("-= Operator Example, Value of c = %d\n", c );
```

```
        c *=  a;
        printf("*= Operator Example, Value of c = %d\n", c );
        c /=  a;
        printf("/= Operator Example, Value of c = %d\n", c );
        c  = 200;
        c %=  a;
        printf("%= Operator Example, Value of c = %d\n", c );
        return 0;
}
```

Output:

```
        = Operator Example, Value of c = 21
        += Operator Example, Value of c = 42
        -= Operator Example, Value of c = 21
        *= Operator Example, Value of c = 441
        /= Operator Example, Value of c = 21
        = Operator Example, Value of c = 11
```

Ternary operator/Conditional operator

- As conditional operator works on three operands, so it is also known as the ternary operator.
- The behaviour of the conditional operator is similar to the 'if-else' statement as 'if-else' statement is also a decision-making statement.

Syntax:

```
        Expression1? expression2: expression3;
```

Example:

```
#include <stdio.h>
int main()
{
   int age;  // variable declaration
   printf("Enter your age: ");
   scanf("%d",&age);   // taking user input for age variable
   // conditional operator
   (age>=18)? (printf("Eligible for voting")) : (printf("Not eligible for voting"));
    return 0;
}
```

Output:

```
        Enter your age: 28
        Eligible for voting
```

Bitwise operator

- The bitwise operators are the operators used to perform the operations on the data at the bit-level.

Operator	Description
&	Binary AND Operator copies a bit to the result if it exists in both operands.
|	Binary OR Operator copies a bit if it exists in either operand.
^	Binary XOR Operator copies the bit if it is set in one operand but not both.
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.

Example:

```
#include <stdio.h>
int main() {
        unsigned int a = 60;    /* 60 = 0011 1100 */
        unsigned int b = 13;    /* 13 = 0000 1101 */
        int c = 0;
        c = a & b;      /* 12 = 0000 1100 */
        printf("Value of c is %d\n", c );
        c = a | b;      /* 61 = 0011 1101 */
        printf("Value of c is %d\n", c );
        c = a ^ b;      /* 49 = 0011 0001 */
        printf("Value of c is %d\n", c );
        c = ~a;         /*-61 = 1100 0011 */
        printf("Value of c is %d\n", c );
        c = a << 2;     /* 240 = 1111 0000 */
        printf("Value of c is %d\n", c );
        c = a >> 2;     /* 15 = 0000 1111 */
        printf("Value of c is %d\n", c );
}
```

Output:

```
        Value of c is 12
        Value of c is 61
        Value of c is 49
        Value of c is -61
        Value of c is 240
        Value of c is 15
```

Increment or Decrement operator

Increment Operator

- Increment Operators are the unary operators used to increment or add 1 to the operand value.
- The Increment operand is denoted by the double plus symbol (++).
- It has two types, Pre Increment and Post Increment Operators.

1. *Pre-increment Operator*

- The pre-increment operator is used to increase the original value of the operand by 1 before assigning it to the expression.

Syntax:

```
X = ++A;
```

2. *Post increment Operator*

- The post-increment operator is used to increment the original value of the operand by 1 after assigning it to the expression.

Syntax

```
X = A++;
```

Decrement Operator

- Decrement Operator is the unary operator, which is used to decrease the original value of the operand by 1.
- The decrement operator is represented as the double minus symbol (--).
- It has two types, Pre Decrement and Post Decrement operators.

1. *Pre Decrement Operator*

- The Pre Decrement Operator decreases the operand value by 1 before assigning it to the mathematical expression.
- In other words, the original value of the operand is first decreases, and then a new value is assigned to the other variable.

Syntax

```
B = --A;
```

2. *Post decrement Operator*

- Post decrement operator is used to decrease the original value of the operand by 1 after assigning to the expression.

Syntax

```
B = A--;
```

Example:

```
#include <stdio.h>
int main ()
{
        int a = 6;
        int b;
        b = ++a;
        printf("Example of pre-increment: %d", b);
        b = a++;
        printf("\nExample of post-increment: %d", b);
        a = 4;
        b = --a;
        printf("\nExample of pre-increment: %d", b);
        b = a--;
        printf("\nExample of post-increment: %d", b);
        return 0;
}
```

Output:

```
        Example of pre-increment: 7
        Example of post-increment: 7
        Example of pre-increment: 3
        Example of post-increment: 3
```

Special Operators (sizeof and comma)

sizeof() operator

- The sizeof operator is the most common operator in C.
- It is a compile-time unary operator and used to compute the size of its operand.
- It returns the size of a variable.
- It can be applied to any data type, float type, pointer type variables.

Example:

```
#include <stdio.h>
int main() {
        int a = 16;
        printf("Size of variable a : %d",sizeof(a));
        printf("\nSize of int data type : %d",sizeof(int));
        printf("\nSize of char data type : %d",sizeof(char));
        printf("\nSize of float data type : %d",sizeof(float));
        printf("\nSize of double data type : %d",sizeof(double));
        return 0;
}
```

Output:

```
        Size of variable a : 4
        Size of int data type : 4
        Size of char data type : 1
        Size of float data type : 4
        Size of double data type : 8
```

Comma as an Operator

- The comma operator is a binary operator that evaluates its first operand, and then discards the result, then evaluates the second operand and returns the value.
- The comma operator has the lowest precedence in C.

Example:

```
#include<stdio.h>
main() {
        int a = 50;
        int b = (a++, ++a);
        printf("%d", b);
}
```

Output

```
        52
```

Evaluation of Expression, Operator Precedence and Associativity

- Expressions are evaluated by the 'C' compiler based on precedence and associativity rules.
- If an expression contains different priority operators, then the precedence rules are considered.
- If an expression contains same priority, then associativity rules are considered i.e. left right (or right to left).

Example:

```
#include <stdio.h>
int main() {
        int a = 20;
        int b = 10;
        int c = 15;
        int d = 5;
        int e;
        e = (a + b) * c / d; // ( 30 * 15 ) / 5
        printf("Value of (a + b) * c / d is : %d\n", e );
        e = ((a + b) * c) / d; // (30 * 15 ) / 5
        printf("Value of ((a + b) * c) / d is : %d\n" , e );
        e = (a + b) * (c / d); // (30) * (15/5)
        printf("Value of (a + b) * (c / d) is : %d\n", e );
        e = a + (b * c) / d; // 20 + (150/5)
        printf("Value of a + (b * c) / d is : %d\n" , e );
        return 0;
}
```

Output:

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is : 90
Value of (a + b) * (c / d) is : 90
Value of a + (b * c) / d is : 50
```

Exercise

1. Short Note
    a. Conditional Operator [TU 2079]
    b. Bitwise Operator [TU 2078]
    c. Operator precedence and associativity [TU 2077]
2. Discuss different logical operation in detail. [TU 2075]
3. Discuss increment and decrement operators with example. [TU 2074]