Operators and Expressions

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

Types of Operators:

- **Arithmetic Operators**
  - + : 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 integer value by one
  - -- : Decrement operator decreases integer value by one

- **Relational Operators**
  - == : Checks if the values of two operands are equal or not, if yes then condition becomes true.
  - != : Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
  - > : Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
  - < : Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
  - >= : Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
  - <= : Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

- **Logical Operators**

- && : Called Logical AND operator. If both the operands are non zero then condition becomes true.
- || : Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.
- ! : Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

- **Bitwise Operators**
  - & : 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 Ones 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.

- **Assignment Operators**
  - = : Simple assignment operator, Assigns values from right side operands to left side operand
  - += : Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand
  - - = : Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand
  - *= : Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand
  - / = : Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand

- % = : Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand
- <<= : Left shift AND assignment operator
- >>= : Right shift AND assignment operator
- & = : Bitwise AND assignment operator
- ^= : bitwise exclusive OR and assignment operator
- |= : bitwise inclusive OR and assignment operator

- **Ternary Operator**
  - C# includes a decision-making operator ?: which is called the conditional operator or ternary operator.
  - It is the short form of the if else conditions.
  - condition ? statement 1 : statement 2

Loop Iteration

- **for loop**
  - A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
  - Syntax:

```
for ( init; condition; increment ) {
        statement(s);
}
```

- **Foreach**
  - The foreach loop is used to iterate over the elements of the collection. The collection may be an array or a list.
  - It executes for each element present in the array.
  - Syntax:

```
for(dataType varName in collection){
```

```
        statement(s);
    }
```

- **While**
  - A while loop statement in C# repeatedly executes a target statement as long as a given condition is true.
  - Syntax:

```
while(condition) {
        statement(s);
}
```

- **do..while**
  - Unlike for and while loops, which test the loop condition at the start of the loop, the do...while loop checks its condition at the end of the loop.
  - A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.
  - Syntax:

```
do {
        statement(s);
} while( condition );
```

- **Break**
  - When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
  - It can be used to terminate a case in the switch statement.
  - If you are using nested loops, the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.
  - Syntax:

```
break;
```

- **Continue**
    - The continue statement works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.
    - For the for loop, continue statement causes the conditional test and increment portions of the loop to execute.
    - Syntax:

      Continue;

Understanding Arrays

- An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.
- Declaring Arrays: datatype[] arrayName;
- Initializing an Array : double[] balance = new double[10];
- Assigning Values to an Array :
    - double[] balance = new double[10];
      balance[0] = 4500.0;
    - double[] balance = { 2340.0, 4523.69, 3421.0};
    - int [] marks = new int[5] { 99, 98, 92, 97, 95};
    - int [] marks = new int[] { 99, 98, 92, 97, 95};

- Accessing Array Elements : double salary = balance[9];

Defining and Calling Methods

- A method is a group of statements that together perform a task. Every C# program has at least one class with a method named Main.
- To use a method, you need to – Define the method , Call the method
- When you define a method, you basically declare the elements of its structure.
- Syntax:

```
<Access Specifier> <Return Type> <Method Name>(Parameter List) {
        Method Body
}
```

- Access Specifier − This determines the visibility of a variable or a method from another class.
- Return type − A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.
- Method name − Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.
- Parameter list − Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- Method body − This contains the set of instructions needed to complete the required activity.

different type of parameters in method :

- **Value type:**
    - This is the default mechanism for passing parameters to a method. In this mechanism, when a method is called, a new storage location is created for each value parameter.

- The values of the actual parameters are copied into them. Hence, the changes made to the parameter inside the method have no effect on the argument .
- Syntax :
  <Access Specifier> <Return Type> <Method Name>
  (<Data Type>< parameter Name>,..) {
  Method Body
  }

- **Reference type:**
  - A reference parameter is a reference to a memory location of a variable. When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters.
  - The reference parameters represent the same memory location as the actual parameters that are supplied to the method.
  - You can declare the reference parameters using the ref keyword.
  - Syntax:
    <Access Specifier> <Return Type> <Method Name>
    (**ref** <Data Type>< parameter Name>,..) {
    Method Body
    }
- **Optional**
  - As the name suggests optional parameters are not compulsory parameters, they are optional. It helps to exclude arguments for some parameters.
  - Each and every optional parameter contains a default value which is the part of its definition.
  - The optional parameters are always defined at the end of the parameter list.
  - Syntax:
    <Access Specifier> <Return Type> <Method Name>
    (<Data Type>< parameter Name**>,..,<Data Type>< parameter Name>=<value>,..**) {

Method Body

}

Working with strings

- strings are array of characters, However, more common practice is to use the string keyword to declare a string variable. The string keyword is an alias for the System.String class.
- The *System.String* class is immutable, i.e once created its state cannot be altered.
- It allows empty strings. Empty strings are the valid instance of String objects that contain zero characters.
- It also supports searching strings, comparison of string, testing of equality, modifying the string, normalization of string, copying of strings, etc.
- Properties of the String Class :
- Chars - Gets the Char object at a specified position in the current String object.
- Length - Gets the number of characters in the current String object.
- Syntax : String <name> = <value>

- **Methods of the String Class:**
  - Compare(String, String) : It is used to compares two specified String objects. It returns an integer that indicates their relative position in the sort order.
  - Concat(String, String): It is used to concatenate two specified instances of String.
  - Contains(String) : It is used to return a value indicating whether a specified substring occurs within this string.
  - Equals(String, String) : It is used to determine that two specified String objects have the same value.
  - IndexOf(String) : It is used to report the zero-based index of the first occurrence of the specified string in this instance.

- Insert(Int32, String) : It is used to return a new string in which a specified string is inserted at a specified index position.
- Remove(Int32) : It is used to return a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted.
- Replace(String, String) : It is used to return a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.
- Split(Char[]) : It is used to split a string into substrings that are based on the characters in an array.
- Substring(Int32) : It is used to retrieve a substring from this instance. The substring starts at a specified character position and continues to the end of the string.
- Trim() : It is used to remove all leading and trailing white-space characters from the current String object.

Datetime class study

- We used the DateTime when there is a need to work with the dates and times
- We can format the date and time in different formats by the properties and methods of the DateTime.
- **Properties of DateTime :**
  - DayOfWeek : We can get the name of the day from the week with the help of the DayOfWeek property.
  - DayOfYear : To get the day of the year, we will use DayOfYear property.
  - TimeOfDay : To get time in a DateTime, we use TimeOfDay property.
  - Today : Today property will return the object of the DateTime, which is having today's value. The value of the time is 12:00:00
  - Now : The Now property will return the DateTime object, which is having the current date and time.