What is operator keyword?

The operator keyword is used to define overloaded operators for user-defined types, allowing them to behave like built-in types. This enables us to customize how operators such as +, -, *, /, ==, !=, <<, >>, and others work with objects of our own classes.

Rules of Operator Overloading

1. Syntax and Declaration:

    - Use the operator keyword followed by the operator to be overloaded.
    - Operators can be overloaded as member functions or non-member (global) functions.
    - The syntax for a member function is return_type operator@(parameters).
    - The syntax for a global function is return_type operator@(parameters).

2. Member vs. Non-member Functions:

    - Unary operators should generally be overloaded as member functions.
    - Binary operators can be overloaded as either member or non-member functions. If the left operand should be of the type of the class (or convertible to it), a member function is appropriate. If the left operand might be of a different type, a non-member function should be used.

3. Non-overloadable Operators:

    - Some operators cannot be overloaded: . (member access), .* (pointer-to-member access), :: (scope resolution), ?: (ternary conditional), sizeof (size of object), typeid (type identification), and alignof (alignment requirement).

4. Preserving Operator Precedence:

    - We cannot change the precedence and associativity of operators through overloading. They retain their original precedence and associativity.

5. Access to Private and Protected Members:

    - Operator overloads, whether as member or non-member functions, have access to the private and protected members of the class.

6. Return Type:

    - The return type of an operator overload can be anything, but it should make logical sense. For example, the + operator typically returns a new instance of the class type.

7. Function Overloading:

    - Overloaded operators can be overloaded themselves, provided they have different parameter lists.