Dead code elimination is an optimization technique used in compiler design to remove portions of code that do not contribute to the final output of a program.

Dead code refers to sections of code that are never executed during the program's runtime, either because they are unreachable or their results are never used.

Dead code elimination helps improve the efficiency and performance of the compiled program by reducing unnecessary computations and reducing code size.

For example,

```
z = x*y;
a = x;  // dead code
w = x*y+6;
```

After dead code elimination above program become,

```
z = x*y;
w = x*y+6;
```

Overview of dead code elimination:

1. Definition: Dead code elimination is the process of identifying and removing code that has no impact on the program's final output or result.
2. Unreachable Code: Dead code can occur when there are portions of code that are impossible to execute based on program logic or control flow. For example, code

following a return statement, code inside an if statement that is never true, or code after an unconditional branch.

3. Unused Results: Dead code can also include computations or assignments that are never used or accessed later in the program. For example, assigning a value to a variable that is never read or performing a calculation that is never utilized.
4. Static Analysis: Dead code elimination is typically performed using static analysis techniques during the compilation process. The compiler analyzes the control flow and data flow of the program to identify code that is dead or unreachable.
5. Marking and Removal: During the analysis, the compiler marks the sections of code that are determined to be dead. In subsequent stages, the marked code is removed from the final generated code or optimized out.
6. Benefits: Dead code elimination offers several benefits, including:
    - Improved performance: Removing dead code reduces unnecessary computations, leading to faster execution times.
    - Reduced code size: Eliminating dead code reduces the size of the compiled program, resulting in smaller executables and reduced memory usage.
    - Simplified maintenance: Removing dead code improves code readability and maintainability by eliminating irrelevant or confusing sections.
7. Limitations: Dead code elimination has certain limitations and challenges. It relies on accurate and precise analysis, which can be complex for programs with dynamic control flow or indirect function calls. Additionally, code that appears to be dead during static analysis might be conditionally executed in specific runtime scenarios, making it challenging to determine its actual reachability.