The process of Code Optimization Involves

* Eliminating the unwanted code lines
* Rearranging the statements of the code

The optimized code has the following advantages-

* Optimized code has faster execution speed.
* Optimized code utilizes the memory efficiently.
* Optimized code gives better performance.

**Important code optimization techniques are-**



1. **Compile Time Evaluation**
   1. **Constant Folding**
      1. involves folding the constants.
      2. The expressions that contain the operands having constant values at compile time are evaluated.
      3. Those expressions are then replaced with their respective results.

**Example :** area =22/7\*radius\*radius;

**Optimized Code :** area=3.142\*radius\*radius

* 1. **Constant Propagation**
     1. If some variable has been assigned some constant value, then it replaces that variable with its constant value in the further program during compilation.
     2. The condition is that the value of variable must not get alter in between.

**Example :** pi=3.142; radius=10;

Area =pi\*radius\*radius;

**Optimized Code :** Area=3.142\*10\*10;

1. **Common Subexpression Elimination**
   1. It involves eliminating the common sub expressions.
   2. The redundant expressions are eliminated to avoid their re-computation.
   3. The already computed result is used in the further program when required.

|  |  |
| --- | --- |
| **Code Before Optimization** | **Code After Optimization** |
| S1 = 4 x i  S2 = a[S1]  S3 = 4 x j  S4 = 4 x i **// Redundant Expression**  S5 = n  S6 = b[S4] + S5 | S1 = 4 x i  S2 = a[S1]  S3 = 4 x j  S5 = n  S6 = b[S1] + S5 |

1. **Code Movement**
   1. It involves movement of the code.
   2. The code present inside the loop is moved out if it does not matter whether it is present inside or outside.
   3. Such a code unnecessarily gets execute again and again with each iteration of the loop.
   4. This leads to the wastage of time at run time.

|  |  |
| --- | --- |
| **Code Before Optimization** | **Code After Optimization** |
| for ( int j = 0 ; j < n ; j ++)  {  x = y + z ;  a[j] = 6 \* j;  } | x = y + z ;  for ( int j = 0 ; j < n ; j ++)  {  a[j] = 6 \* j;  } |

1. **Dead Code Elimination**
   1. It involves eliminating the dead code.
   2. The statements of the code which either never executes or are unreachable or their output is never used are eliminated.

|  |  |
| --- | --- |
| **Code Before Optimization** | **Code After Optimization** |
| i = 0 ;  if (i == 1)  {  a = x + 5 ;  } | i = 0 ; |

1. **Strength Reduction**
   1. It involves reducing the strength of expressions.
   2. This technique replaces the expensive and costly operators with the simple and cheaper ones.

|  |  |
| --- | --- |
| **Code Before Optimization** | **Code After Optimization** |
| B = A x 2 | B = A + A |