Code optimization

An important phase of compiler related to improving the quality of codes by removing unnecessary lines as well as rearranging the statements of the code

Why it is needed (The Purpose)

- This makes code run faster and hence optimized code gives better performance. i.e. code optimization allows consumption of fewer resources. (i.e. CPU, Memory), which results in faster running machine code.

- Optimized code also uses memory efficiently.

Objective

- The optimization must be correct, it must not, in any way, change the meaning of the program.

- Optimization should increase the speed and performance of the program.

- The compilation time must be kept reasonable.

- The optimization process should not delay the overall compiling process.

Which Code to optimise

Intermediate codes generated by compiler. After optimization, intermediate code are transformed and improved

Two types of code optimization techniques:

1. **Machine independent optimization** — aims to improve the intermediate code (this code does not involve any CPU registers or absolute memory locations). It aims to get a better target code as output.

2**. Machine dependent optimization-**

- During the object code generation Phase
- The code is transformed according to target machine architecture. It involves CPU registers and may have absolute memory references rather than relative references.

Level of optimization- Local, Global and Loop

Local- applied locally to a specific part of program. Other parts of program are unaffected. Usually appied to one basic block (sequence of intermediate code, the flow different possible paths of various basic blocks of a program is called control flow graph)

Global- it is applied across all basic blocks (i.e. to entire program). Yields better results but requires sophisticated analysis (Data flow Analysis)

Loop- is usually the main target for any optimisation techniques due to their structure and optimisation possibility. Loop optimisation may be at local or global level, However it usually involves more than one basic blocks

Popular techniques

1. Common subexpression elimination

2. Dead code elimination (code never executed or their output is never used)

3. Code movement

4. ***Loop invariant*** and ***induction variables*** optimisation (applied to loos optimisation)

4. Strength reduction (replacing expensive operator with cheaper operator)

a = b * 2 → a = b+b

5. Compile Time evaluation (Constant folding & Constant Propagation)

Constant folding- a computation ***a+ 10*3*** will be computed as ***a+30*** at compile time