Front end and back end of the compiler

Front end and back end is the collection of phases of compiler.

Front End :

1. Lexical Analysis,
2. Syntax Analysis,
3. Semantic Analysis,
4. Intermediate Code,
5. Some amount of Code Optimization.

Back End:

1. Code Optimization,
2. Code Generation

1. Lexical Analysis: It takes source program as input and produce tokens.

What is lexical token ?

Ans. A unit in the grammar of the programming language.

Examples:

Type token = (id, number)

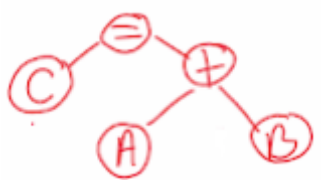
Punctuation token = (if, return)

2. Syntax Analysis: It takes output of lexical analysis as input and produces tree as output.

For example: Output of Lexical analysis = C = A + B

Input of syntax analysis = C = A + B

Output of syntax analysis =



3. Semantic Analysis : It takes output of syntax analysis as input and produces a tree with type information as output.
It checks for sematic errors.
For example: Output of syntax analysis

Input of semantic analysis =

Output of semantic analysis =

4. Intermediate code: It takes output of semantic analysis as input and produces intermediate code as output.

Disadvantages of Front End:
1. Requires large amount of memory to store tokens and trees.
2. Data move from one memory to another which makes it very slow.

Function of Front End:
1. Determine validity of source code.
2. Determine content of source code.
3. Build source code for easy to analyze.

1. Code optimization : It is the process to modify the program to make it more efficient, faster execution, less resources requirements.
Levels of code optimization:
a. Design level
b. Source code level
c. Compile level
d. Assembly level
e. Run time level

2. Code generation : Knowledge of target architecture helps code generation to determine:
a. Where to store result in memory location or registers.
b. Which instruction is better for type conversion.
c. Which addressing mode to use.
For example:
AX = 5, BX = 2, AX+BX
Assembly code :
MOV AX, 5
MOV BX, 2

ADD AX, BX

Related Posts:

1. Introduction to Compiler
2. Analysis and synthesis model of compilation
3. Bootstrapping and Porting
4. Lexical Analyzer: Input Buffering
5. Storage Allocation Strategies
6. Type Checking
7. Specification & Recognition of Tokens
8. LEX
9. Analysis synthesis model of compilation
10. Data structure in CD
11. Register allocation and assignment
12. Loops in flow graphs
13. Dead code elimination
14. Syntax analysis CFGs
15. L-attribute definition
16. Operator precedence parsing
17. Analysis of syntax directed definition
18. Recursive descent parser
19. Function and operator overloading
20. Storage allocation strategies
21. Equivalence of expression in type checking

22. Storage organization
23. Parameter passing
24. Run time environment
25. Type checking
26. Code generation issue in design of code generator
27. Boolean expression
28. Declaration and assignment in intermediate code generation
29. Code optimization
30. Sources of optimization of basic blocks
31. Loop optimization
32. Global data flow analysis
33. Data flow analysis of structure flow graph (SFG)