Foundations of Computer Science – Exercise 6

1. $x_{n+1} = x_n - \frac{x_n^2 - N}{(x_n^2 - N)'} = \frac{1}{2}(x_n - \frac{N}{x_n})$
2.

Address	Command
101	LOAD 1003
102	STORE 1001
103	LOAD 1005
104	JUMPZERO 1007
105	LOAD 1000
106	DIVIDE 1001
107	STORE 1004
108	LOAD 1001
109	SUBTRACT 1004
110	DIVIDE 1006
111	STORE 1001
112	LOAD 1005
113	SUBTRACT 1003
114	STORE 1005
115	LOAD 1001
116	STORE 1002
117	JUMP 103

Address	Value
1000	n
1001	x_n
1002	result
1003	1
1004	N/x_n
1005	5
1006	2

3. a)

Compiler	Interpreter
Execute an entire program at a time	Execute a single line of code at a time
Create intermediate object code	No intermediate object code
Compilation takes place before execution	Compilation and execution are simultaneous
Faster	Slower
Require more memory	Require less memory
Difficult error detection as all errors are displayed after compilation	Easier error detection as errors of each line are displayed one by one
More efficient	Less efficient

b)

Lexical Analysis:

- Identify the lexical units in a source code
- Classify lexical units into classes like constants, reserved words, and enter them in different tables. It will Ignore comments in the source program
- Identify token which is not a part of the language

Syntax Analysis:

- Obtain tokens from the lexical analyzer
- Checks if the expression is syntactically correct or not
- Report all syntax errors
- Construct a hierarchical structure which is known as a parse tree

Semantic Analysis:

- Helps you to store type information gathered and save it in symbol table or syntax tree
- Allows you to perform type checking
- In the case of type mismatch, where there are no exact type correction rules which satisfy the desired operation a semantic error is shown
- Collects type information and checks for type compatibility
- Checks if the source language permits the operands or not

Intermediate Code Generation:

- It should be generated from the semantic representation of the source program
- Holds the values computed during the process of translation
- Helps you to translate the intermediate code into target language
- Allows you to maintain precedence ordering of the source language
- It holds the correct number of operands of the instruction

Code Optimization:

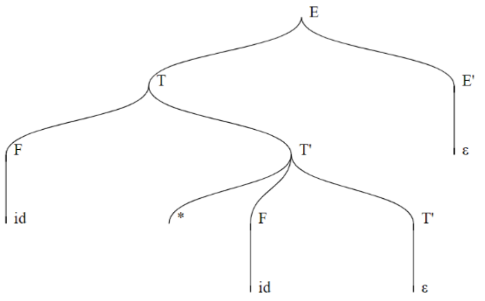
- It helps you to establish a trade-off between execution and compilation speed
- Improves the running time of the target program
- Generates streamlined code still in intermediate representation
- Removing unreachable code and getting rid of unused variables
- Removing statements which are not altered from the loop

Code Generation: Code generation is the last and final phase of a compiler. It gets inputs from code optimization phases and produces the page code or object code as a result. The objective of this phase is to allocate storage and generate relocatable machine code.

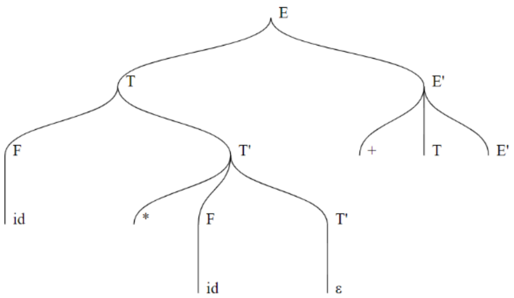
c) Lexical Analysis: Store in symbol table
Syntax Analysis: Store in parse tree.
Code generation: Store in code.

4.

a) E→TE’→FT’E’→**id**T’E’→**id**E’→**id**+TE’→**id**+FT’→**id**+**id***FT’→**id**+**id***id**
b) Yes as the sentence was parsed based on grammar provided by the parsing table.
c) **id * id**



d) Parsing failed:



5. a) Digit$^+$ Other$^+$
   b) Tokens that do not follow the Digit – Other – Digit – Other … pattern.