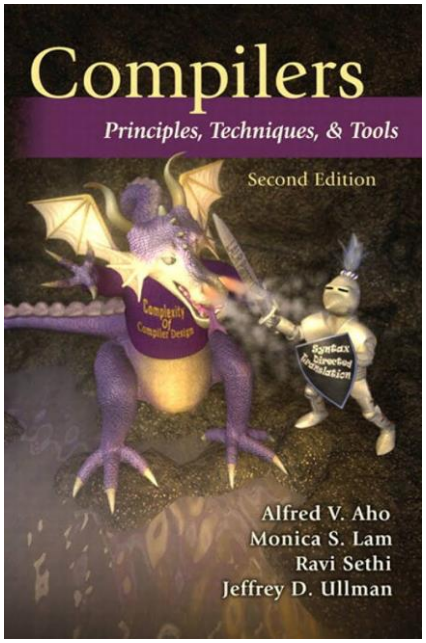
Compiler

LEC 1

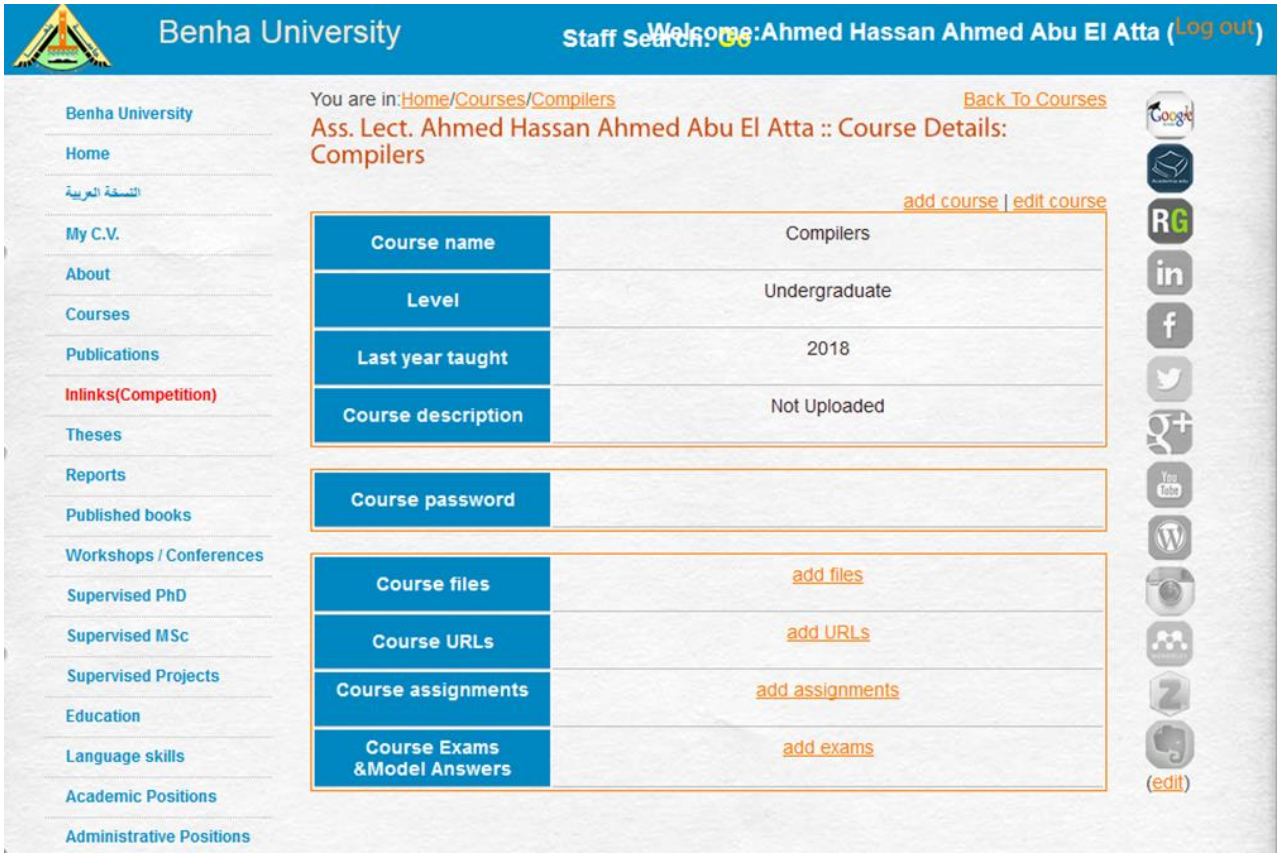
Book

Compilers: Principles, Techniques, and Tools is a computer science textbook by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman about compiler construction.



PowerPoint

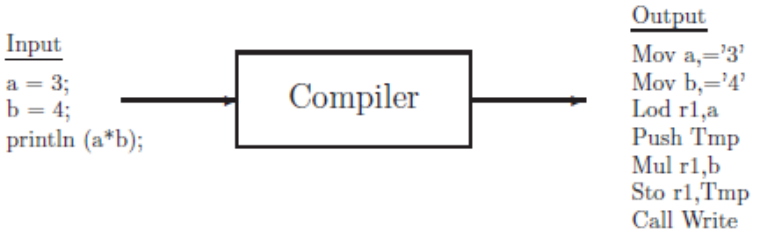
http://www.bu.edu.eg/staff/ahmedaboalatah14-courses/14779



Compiler & Interpreter

Compiler is a software computer program. It reads program in source language and translates it into an equivalent program in target language.

Interpreter executes each command language line by line.



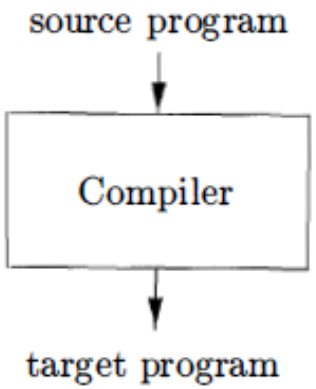


Figure 1.1: A compiler

Figure 1.2: Running the target program

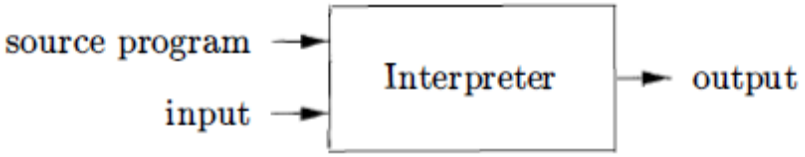
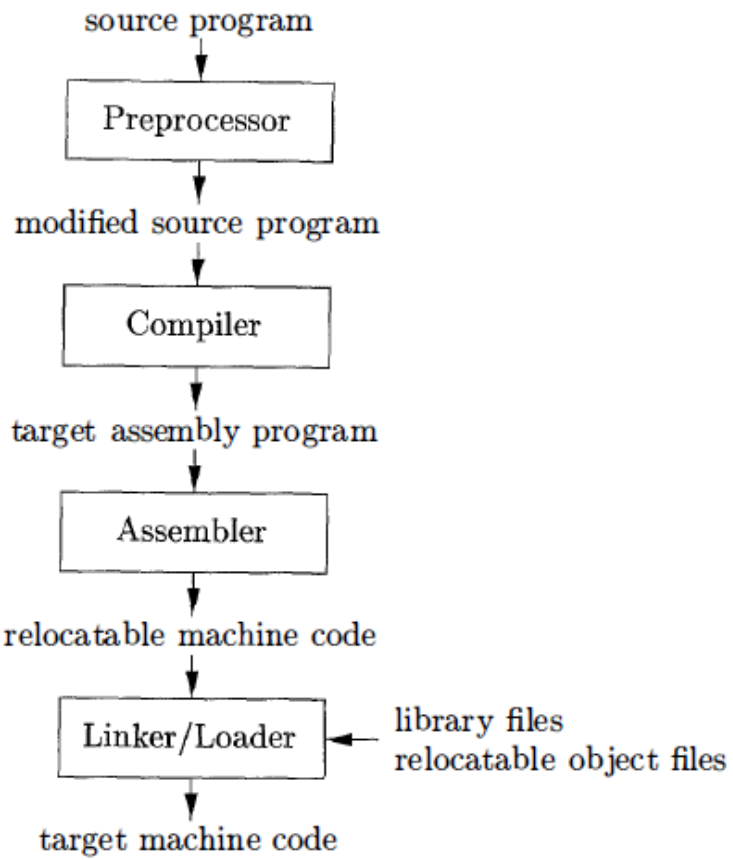


Figure 1.3: An interpreter

A language-processing system



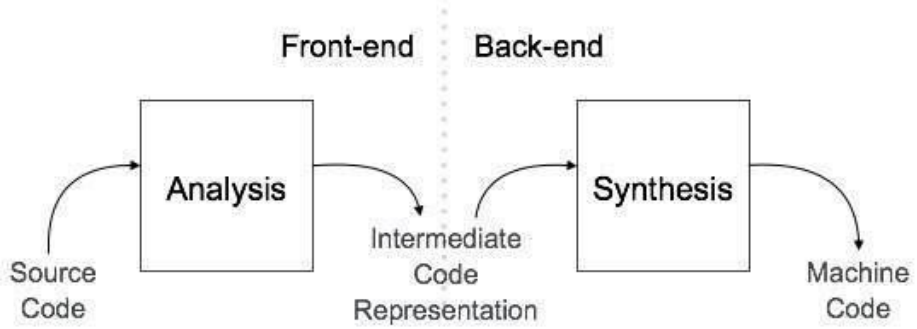
A compiler can broadly be divided into two phases based on the way they compile.

Analysis Phase

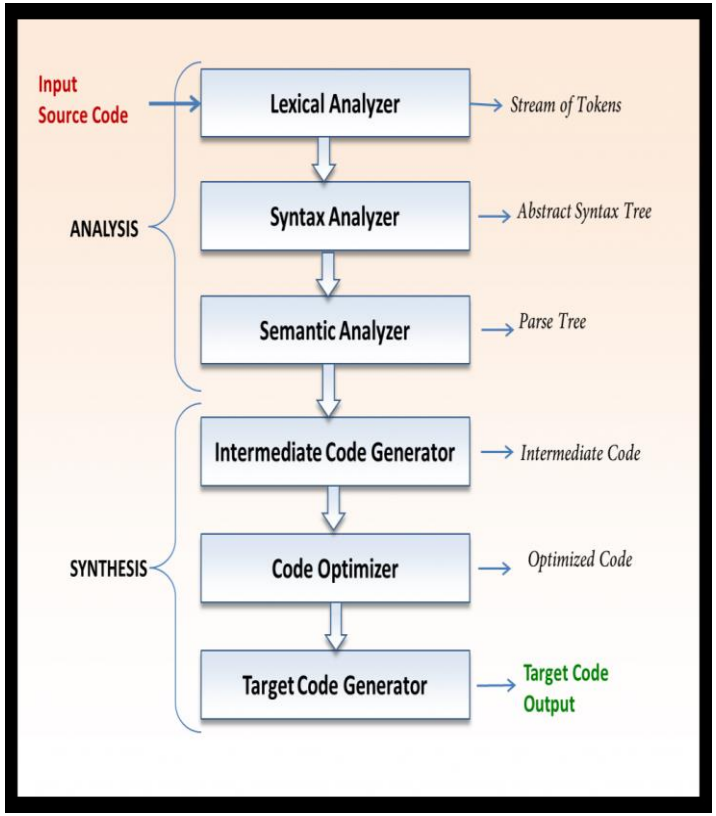
- Known as the front-end of the compiler, the analysis phase of the compiler reads the source program, divides it into core parts and then checks for lexical, grammar and syntax errors. The analysis phase generates an intermediate representation of the source program and symbol table, which should be fed to the Synthesis phase as input.

Synthesis Phase

- Known as the back-end of the compiler, the synthesis phase generates the target program with the help of intermediate source code representation and symbol table.



Phases of a Compiler



Lexical Analysis

- This phase scans the source code as a stream of characters and converts it into meaningful lexemes (tokens).

Syntax Analysis

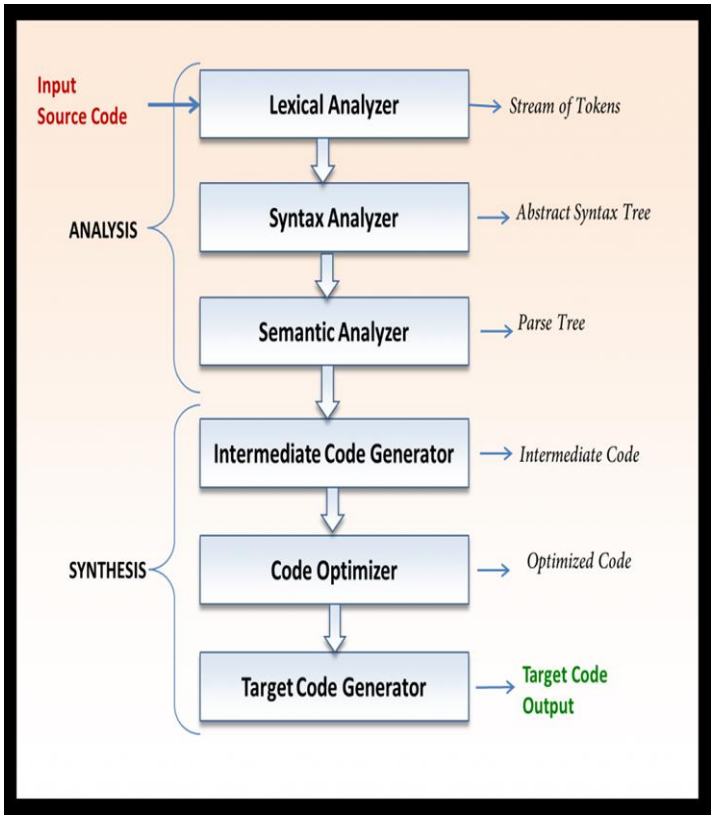
- It takes the token produced by lexical analysis as input and generates a parse tree (or syntax tree). In this phase, token arrangements are checked against the source code grammar.

Semantic Analysis

- Semantic analysis checks whether the parse tree constructed follows the rules of language. For example, assignment of values is between compatible data types, and adding string to an integer. Also, the semantic analyzer keeps track of identifiers, their types and expressions; whether identifiers are declared before use or not etc. The semantic analyzer produces an annotated syntax tree as an output.

Intermediate Code Generation

- After semantic analysis the compiler generates an intermediate code of the source code for the target machine. It represents a program for some abstract machine. It is in between the high-level language and the machine language.



Code Optimization

- Optimization can be assumed as something that removes unnecessary code lines, and arranges the sequence of statements in order to speed up the program execution without wasting resources (CPU, memory).

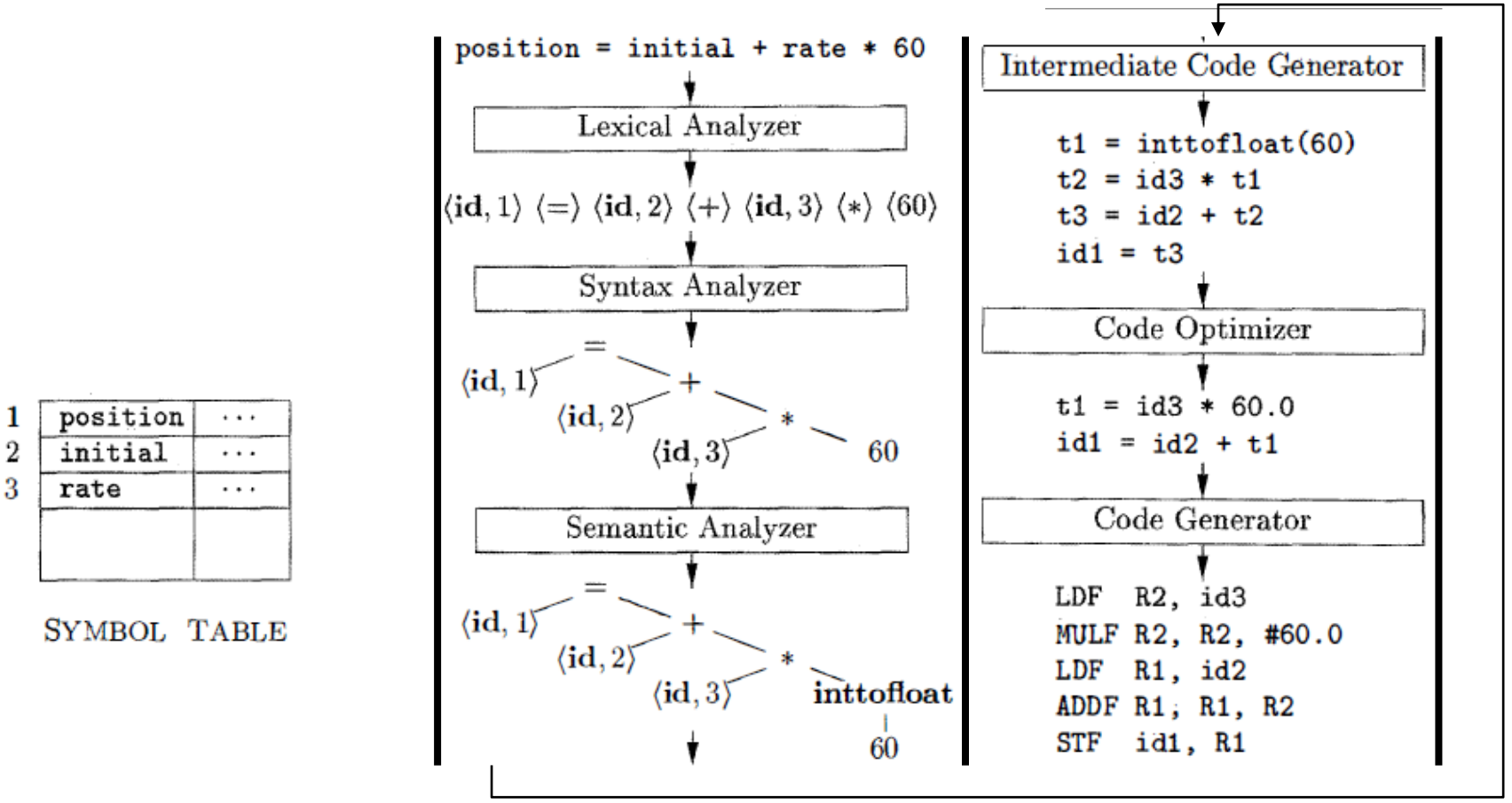
Code Generation

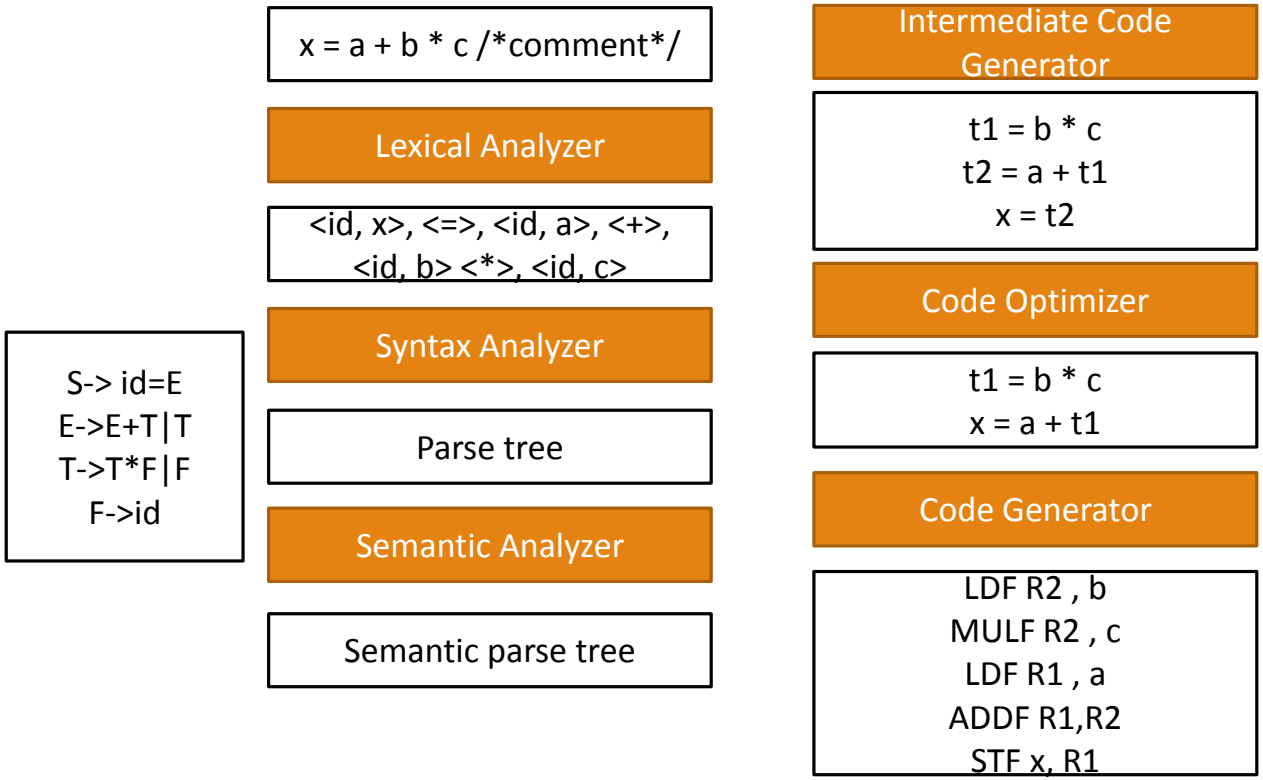
- In this phase, the code generator takes the optimized representation of the intermediate code and maps it to the target machine language.

Symbol Table

- It is a data-structure maintained throughout all the phases of a compiler. All the identifier's names along with their types are stored here. The symbol table makes it easier for the compiler to quickly search the identifier record and retrieve it. The symbol table is also used for scope management.

Phases of a compiler




x = a + b * c /*comment*/
Lexical Analyzer
<id, x>, <=>, <id, a>, <+>, <id, b> <*>, <id, c>
Syntax Analyzer
S-> id=E
E->E+T|T
T->T*F|F
F->id
Parse tree
Semantic Analyzer
Semantic parse tree
Intermediate Code Generator
t1 = b * c
t2 = a + t1
x = t2
Code Optimizer
t1 = b * c
x = a + t1
Code Generator
LDF R2 , b
MULF R2 , c
LDF R1 , a
ADDF R1,R2
STF x, R1

A model of a compiler front end