NAME: IBEGBU EJIAMIKE

MATRIC NUMBER: 17/SCI01/036

COURSE CODE: CSC312

COURSE TITLE: COMPILER CONSTRUCTION

**Lexical Analysis:**

This is the first phase of the compilation process and is handled by the lexical analyzer which is also called the Scanner. In this phase the input source code is scanned and separated into lexical units called tokens. The lexical analyses read the input code character-by-character.

Take an example, the line of code below:

String name = “Sergi Alacron”;

The lexical analyzer would generate the following 7 tokens and entered as 7 records in the Symbol Table:

String

name

=

“

Sergi

Alacron

“

;

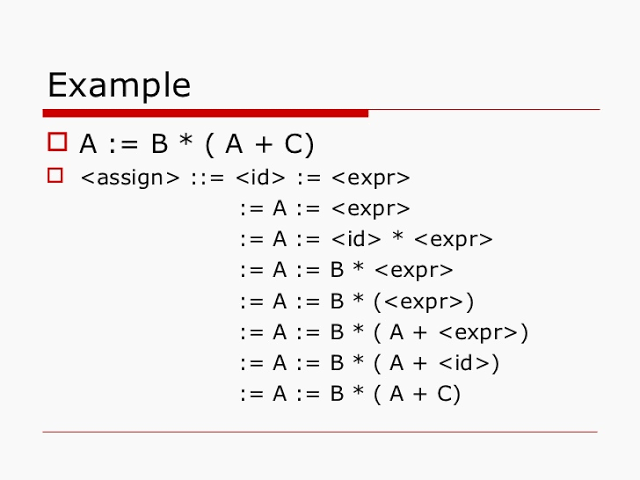
The Symbol table is generated in this phase and populated with tokens generated. A symbol table is typically a data structure that holds a record for each identifier in the source code.

The output of this phase is Stream of Tokens

**Syntax Analysis**

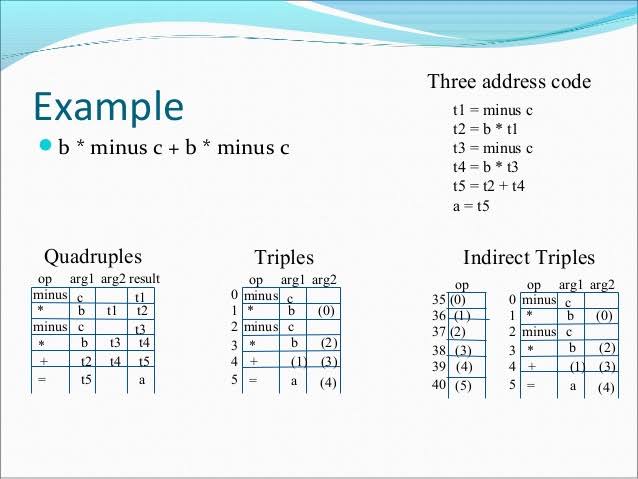
This phase is handled by the syntax analyzer. The stream of tokens generated in the lexical analysis phase is analyzed further to ensure that the input code follows the syntax of the particular language.

Syntax errors are detected in this phase. The output of this phase includes abstract syntax trees



**Semantic Analysis**

Semantic analysis is handled by the Semantic Analyses and has to do with ensuring that the source code follows standard semantic rules.

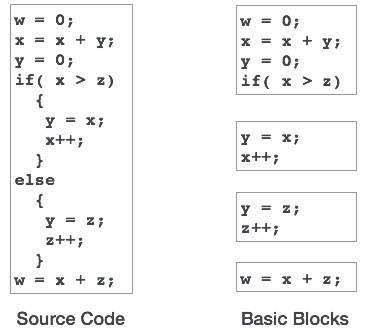
Type Checking is taken care of in this phase. This ensures that the variables are assigned values according to their declaration. So if a variable has been declared as integer and then assigned a float, the error is trapped by the Semantic Analyzer. This phase also identifies chunks of code such as operands and operators of statements in the input code. The output of this phase includes the Parse Tree.

**Intermediate Code Generation**

Intermediate code refers to a code that is somehow between the source code and the target code, an intermediate representation of the input source program. One attribute of an Intermediate Code is ease of translation to target program.

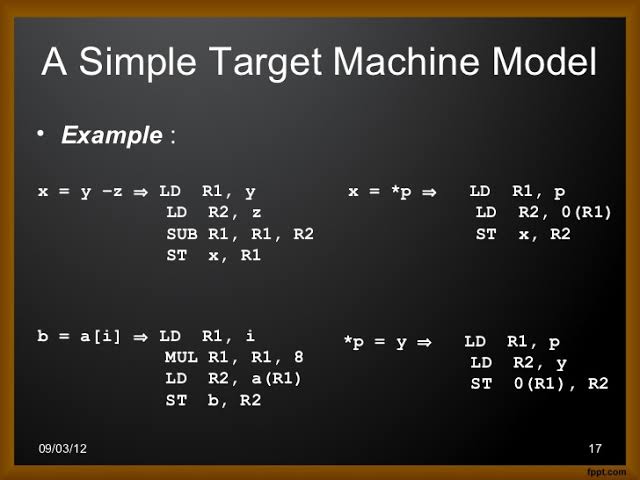
An example would be a c++ programs compiled into cpp Bytecodes (.class files) for the c++ Virtual Machine. This intermediate code can run on any operating system that has the cpp compatibility. One form of intermediate code is the “Three-Address-Code” which resembles an assembly language. The final target code is generated from the intermediate code.

**Code Optimization:**

In Code Optimization, the code is optimized to remove redundant codes and the optimize for efficient memory management as well as improve the speed of execution. The intermediate code ensures that a target code can be generated for any machine enabling portability across different platforms. Output of this phase is the Optimized Code.

**Target Code Generation**

Here the target code is generated for the particular platform. Machine instruction are generated from the optimized intermediate code. Assignment of variables and registers is handled here.

The output of this phase is the target code.