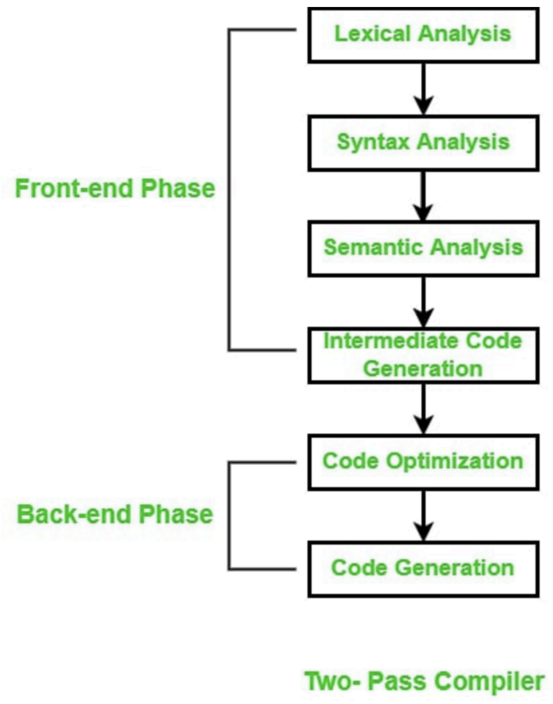
CS 351: Programming Languages
Understanding the Compiler



What is a compiler?

- In simple terms, a compiler is a software that translates complex high level language code into machine code so that the machine is able to execute
- When you press run, the computer attempts to translate into machine code
- If there are any errors along the steps of translating it will stop and print out an error
- If not… it runs smoothly throughout the entire code and is able to execute the commands given, typically giving us an output

1st Step: Lexical Analyzer

- A token includes all of the data types in your code
- In the lexer the program scans for valid tokens and puts them in a list for the syntactic analyzer
- Valid Tokens
    - Int (5, 25, 2343, 233, etc.)
    - float (5.5, 2.2, 5.32, etc.)
    - Constants (MAX, LIMIT, etc.)
    - Identifiers (result, i, j, largest, min, etc.)
    - Operators (+, -, =, etc.)
    - Separators (: , ; etc.)
    - Keywords(int, if, while, etc.)
- If there is an invalid token, the program will stop and return an error

2nd Step: Syntax Analyzer (parser)

- This step anaylyzes all of the tokens given from the previous lexer
- Valid tokens must be coded in the correct order/format
- If the syntax is not correct than the program will return an error
- For example: int a = 5; will pass the parser
- On the other hand int a = ; 5 will not pass
- Even though the second example has valid tokens, it is not in the correct order
- GRAMMAR OF THE CODE

3rd step: Semantic Analyzer(parser)

- Checks type mismatch
    - Number1 = 5
    - Number2 = “hello world”
    - result = Number1 + Number2
    - This would result in an error as we cannot add two different types together
- Checks for undeclared variables
    - Int y=10
    - Int X = z + y
    - This would result in an error because we never declared z
- Checks for correct function calls
    - function(int a, int b) {
      Return a * b }

    - Above is our function
    - Function call: function(5)
    - This would cause an error because the function calls for two arguments and we only gave it one
- There are other examples, but it can be simplified as the secondary level of the parser, checking for incorrect use of the tokens, but not simply by rules or order but instead logic

NOTE: these steps will not be covered in 351 but are here to help you understand the full process

4th step: Intermediate Code Generation

- Here the computer translates the language-specific code to a generic assembly
- This allows the computer to be able to read the code, despite of the language you are using
- Converts to assembly for compiler lanuages

5th step: Code Optimization

- Even though assembly can be read by the computer, it must be in the form of 0’s and 1’s for the computer to execute
- Converts from assembly to binary

6th step: Target Code Generation

- Code is now in binary for the computer to execute