Compilers

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

- Overview

- Compilers
- Scanning
- Parsing
- Semantic analysis
- Intermediate code generation
- Optimization
- Target code generation

Compiling and Running a Program

High-level vs Low-level Languages

- High-level language – Easy for humans to understand. Hides many details. For example, Java and Python.
  - Allow us to focus more on the problem being solved.
- Low-level language – Very detail oriented. Machine-level code is the lowest level. Need to know lots of details about the machine this code is running on.

Compiler

- Compiler - Converts a high-level language to a low-level language.
- For example:

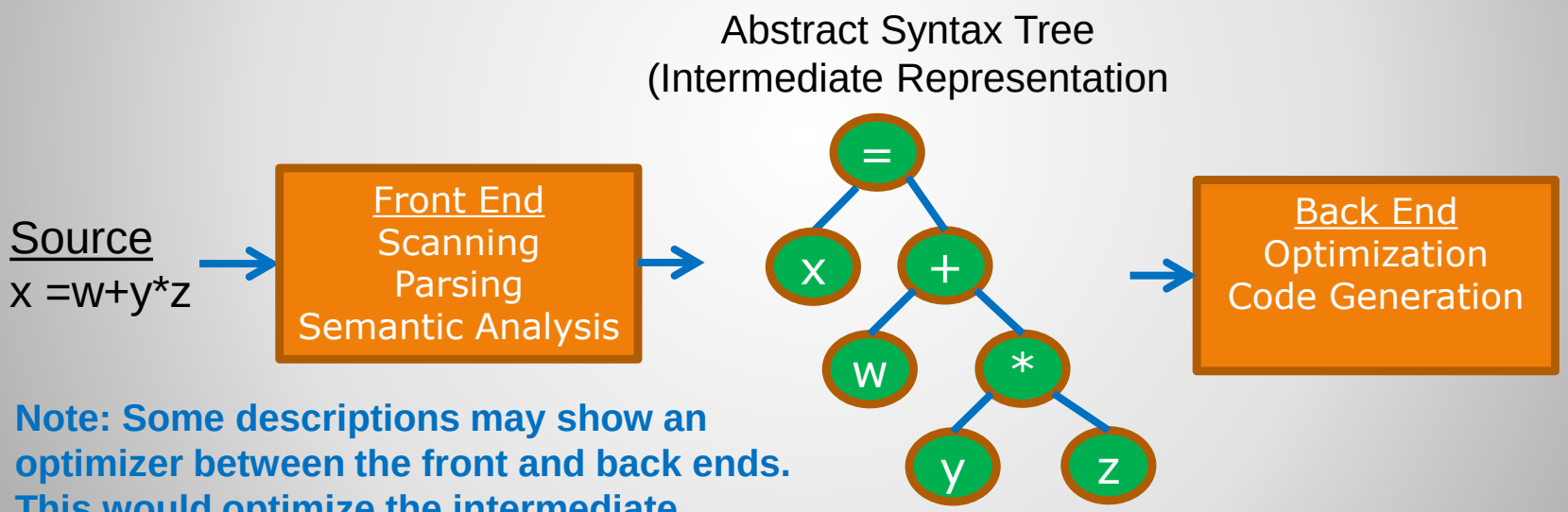

```
// High-level
// Java code
a = b + c
```

```
// Low-level
// Assembler
load r1, a
load r2, b
add r1, r2, r3
store r3, c
```

Compiler Phases: Front and Back Ends

- The compiler phases can be divided into two main parts:
  - Front end
  - Back end
- The front-end phases are:
  - Scanning
  - Parsing
  - Semantic analysis
- The back-end phases are:
  - Optimization
  - Code generation

- Compiler Front End. Responsible for scanning, parsing, and semantic analysis.
  - Input – Source code
  - Output - Abstract syntax tree (intermediate representation of the code)
- Compiler Back End. Responsible for optimization and code generation.
  - Input – Abstract syntax tree
  - Output – Lower-level code (for example Java bytecode, assembly language, machine language)

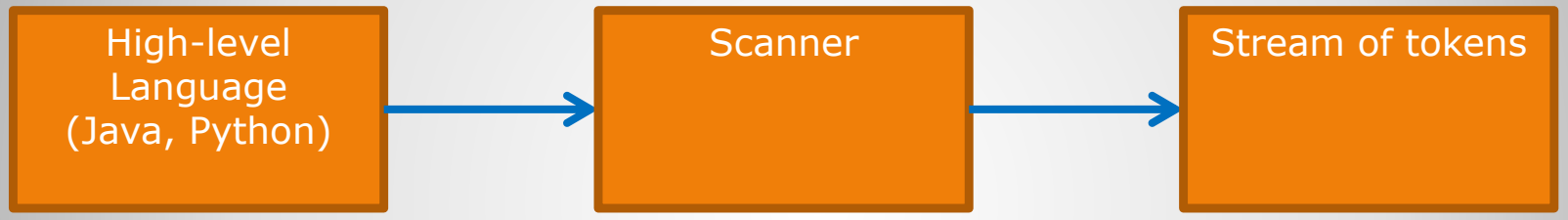


Note: Some descriptions may show an optimizer between the front and back ends. This would optimize the intermediate representation and pass it to the back end.

Compiler Front and Back Ends

Scanning

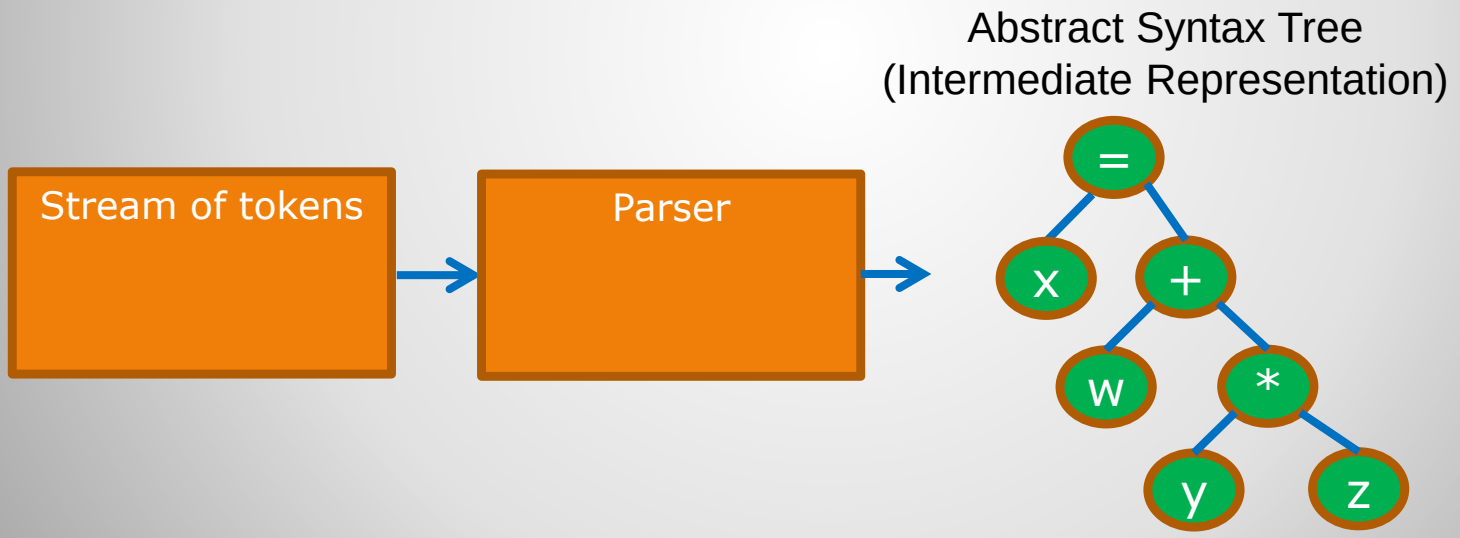
- A program is given to the scanner as input.
- The scanner generates a stream of tokens as output.
- Tokens are like the parts of speech in an English sentence.



Scanner

Reads characters one at a time and groups them together. The groupings are called tokens.

- Parser - The parser takes a stream of tokens as input and produces an abstract syntax tree.
- The parser's input stream of tokens is generated by the scanner during the lexical analysis phase.

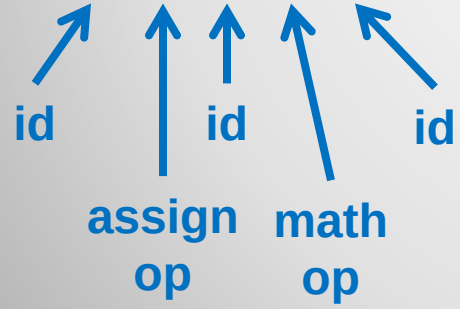


Parsing

Semantic Analysis

- Semantic analysis checks that a statement "makes sense" or has meaning.
- The parser will not check if a statement makes sense (parser checks syntax, only checks if correct according to grammar).
- The types of tokens may be correct, but they may not make sense together.
- For example, data types must match.
- The assignment in the following code is semantically correct:

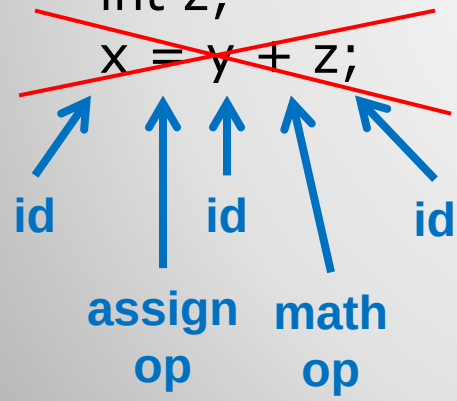
```
int x;
int y;
int z;
x = y + z;
```



Semantic Analysis

- If data types are not compatible in a statement, then it is not semantically correct.
- A statement may be a valid to the parser, but invalid to the semantic analyzer.
- In many languages, a string is on the left side of an assignment and an int is on the right side it will be incorrect.
- The assignment in the following code is NOT semantically correct:

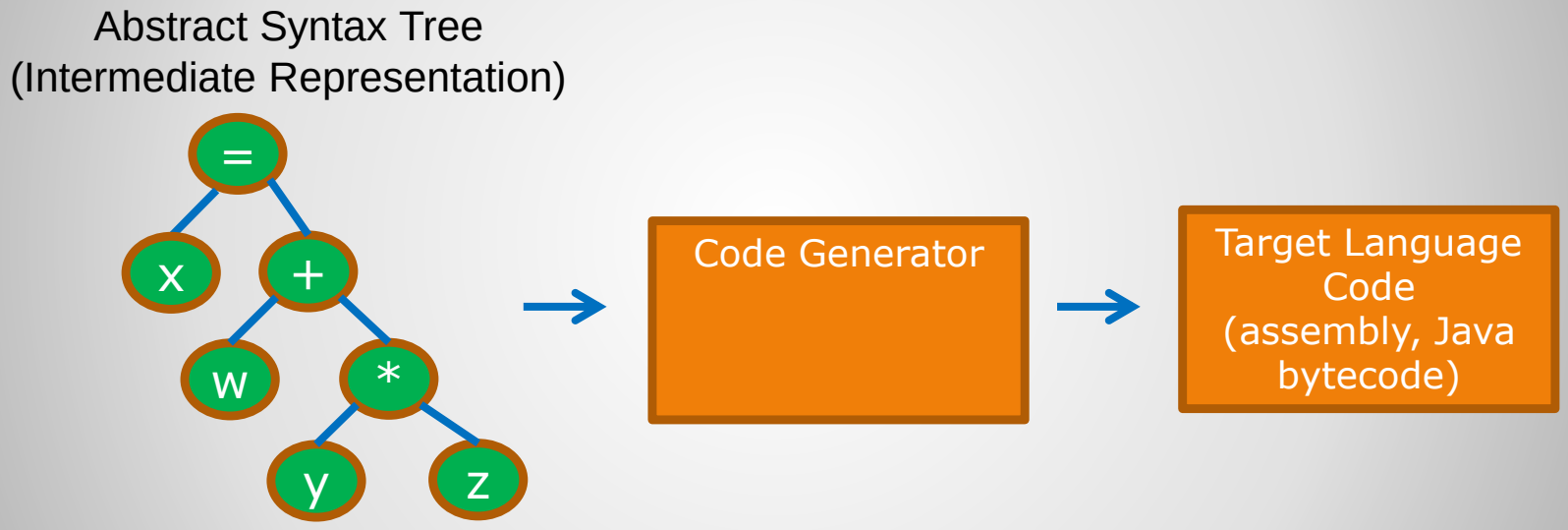
```
String x;
int y;
int z;
x = y + z;
```



Semantically Incorrect!!!

Data types do not match (parser would not identify this error)

- Code Generator - The code generator takes the abstract syntax tree (AST) as input and outputs target language code.
- The code generator traverses the AST and generates code from that traversal.



Code Generation

- **End of Slides**

End of Slides