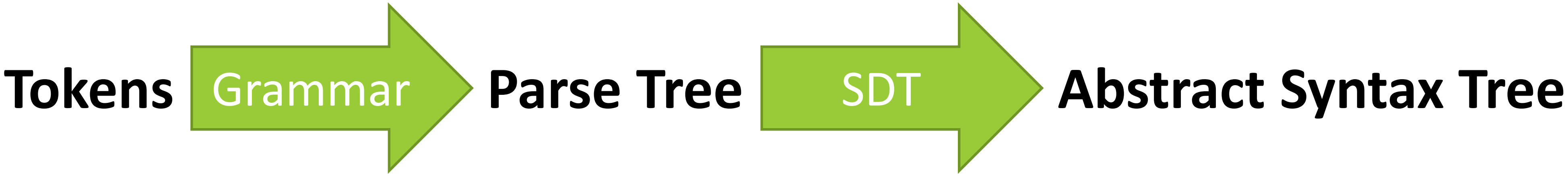
COMP 442/6421 Compiler Design

Instructor: Dr. Joey Paquet paquet@cse.concordia.ca
TA: Zachary Lapointe zachary.lapointe@mail.Concordia.ca

LAB 4 – SYNTAX DERIVED TRANSLATION AND ABSTRACT SYNTAX TREES

Parsing

- Inputs:
  - Token stream/list
  - LL(1) grammar, first sets, follow sets
- Output:
  - Abstract syntax tree
  - Derivation proof



Tokens Grammar **Parse Tree** SDT **Abstract Syntax Tree**

Parsing - Parse Trees

- The parse tree is a representation of the program’s syntactical derivation
- It is not necessarily useful to the compilation process
- We would like to transform it into a more useful representation
  - Abstract Syntax Trees

- The parsing process also represents a program’s syntactical derivation
  - An explicit parse tree is unnecessary
  - Instead, think along the lines of a *virtual* parse tree, from which the AST is instantly generated
    - The parse tree exist only conceptually, represented by the state of the parser

Abstract Syntax Trees

- Represents semantics of parsed program
  - Contains only meaningful constructs
    - Nodes contain semantic information, relevant to later compiler phases
    - No syntax-only constructs: (punctuation, keywords, grammar transformation artifacts, etc.)

- Plan out what nodes and subtrees you'll want in your AST
  - Use these to help design the semantic rules which augment your grammar
  - The lecture slides are a good starting point

- There are multiple ways to construct the tree while parsing
  - Keep in mind the fundamental operations:
    - Making nodes
      - Pulling token information into the node (type, location, lexeme)
    - Grouping nodes
      - Parents, siblings
  - Some examples . . .

Tree Traversal - A Primer

- Tree traversal will be required to effectively use the AST in later phases of the compiler
- Algorithms
  - Pre-order traversal (required for compiler)
  - Post-order traversal (required for compiler)
  - Binary in-order traversal (likely not required)
  - Generalized in-order traversal (likely required for compiler)
  - Euler tour (generalization of the above 3 algorithms)
  - Breadth-first traversal (likely not required)

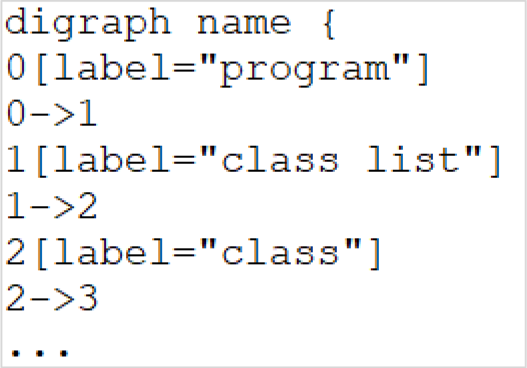
  - These will be covered in more detail next lab

- Design trees with these in mind

AST Visualization - *DOT* and *GVEdit*

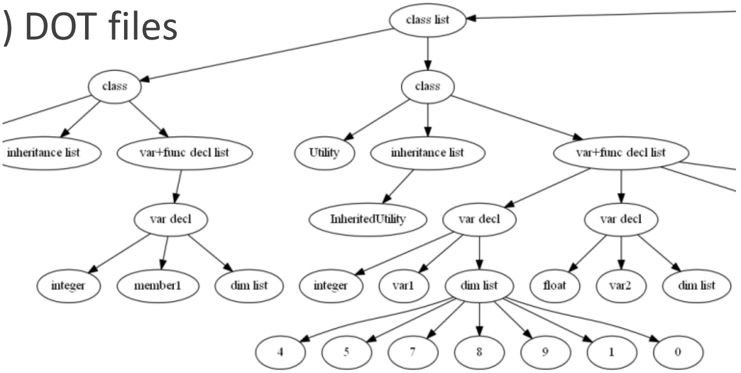
- During the compilation process, the AST exist only in program memory.
  - We would like to inspect it

- DOT file format:
  - An open-source text format for representing graphs
    - Numbered nodes and relations, straightforward to generate while traversing a tree



```
digraph name {
0[label="program"]
0->1
1[label="class list"]
1->2
2[label="class"]
2->3
...
```

- GVEdit:
  - An open-source tool for visualizing (and editing) DOT files



- Some examples . . .