CS 3304
Comparative Languages

Lecture 1: Introduction

17 January 2012

Course Overview

Welcome

- What this course is about?
- What this course is not about?
- What will you learn?
- How will you learn?
  - Read and understand material.
  - Regularly check for announcements and assignments.
  - Interact with instructors and students.
  - Provide feedback.
- Course website: http://scholar.vt.edu/
- Forum website: http://www.piazza.com/

Overview

- **CRN:** 12059
- **Course description:**
  This course in programming language constructs emphasizes the run-time behavior of programs. The languages are studied from two points of view: (1) the fundamental elements of languages and their inclusion in commercially available systems; and, (2) the differences between implementations of common elements in languages. A grade of C or better required in CS prerequisite 3114. I,II.
- **Prerequisites:**
  - CS 3114 (MIN grade of C)

Contacts

- **Instructor**:
  Dr. Denis Gracanin, Associate Professor
  E-mail: gracanin@vt.edu
  Office: KnowledgeWorks II, Room 1135
  Office Hours:
  - Location: LumenHaus (behind Cowgill Hall) or McBryde 122
    Time: Tuesday/Thursday 1pm-3pm
  - Location: KnowledgeWorks II, Room 1135
    Time: By appointment
- **Graduate Teaching Assistant (GTA):**
  Shaimaa Lazem
  E-mail: shlazem@vt.edu
  Office Hours:
  - Location: McBryde 106
    Time: Friday 9am-noon

Textbook and Course Materials

- Michael L. Scott
  Programming Language Pragmatics, Third Edition
  Morgan Kaufmann, 2009
  ISBN-13 978-0-12-374514-9
- Course materials will be posted on the course web site accessible through the VT scholar gateway:
  http://scholar.vt.edu/

Grading

- One midterm exams (closed book): 15%.
- Final exam (comprehensive, closed book): 25%.
- Homeworks: 20% total.
- Programming assignments: 40%.
- Late submission policy: the assignment is first graded as if submitted on time. The points score is then multiplied by the factor *k* based on the time *t* elapsed between the deadline and your submission.
  - $t \leq 24$ hours: $k = 0.9$
  - $24 < t \leq 48$ hours: $k = 0.5$
  - $t > 48$ hours: $k = 0$

Honor System

- Please review The Virginia Tech Undergraduate Honor System, for more information see: http://www.honorsystem.vt.edu/
- All assignments are individual, no collaboration allowed.
- However, you can discuss course topics and related issues as long they do not relate to homework or project specifics. When in doubt, consult us first.
- All code will be checked by a tool that detects similarities and warns about possible plagiarism.

Introduction

Why So Many Programming Languages?

- Evolution: we've learned better ways of doing things over time.
- Socio-economic factors: proprietary interests, commercial advantage.
- Orientation toward special purposes.
- Orientation toward special hardware.
- Diverse ideas about what is pleasant to use.

What Makes a Language Successful?

- Easy to learn (BASIC, Pascal, LOGO, Scheme).
- Easy to express things, easy use once fluent, "powerful" (C, Common Lisp, APL, Algol-68, Perl).
- Easy to implement (BASIC, Forth).
- Possible to compile to very good (fast/small) code (Fortran).
- Backing of a powerful sponsor (COBOL, PL/1, Ada, Visual Basic).
- Wide dissemination at minimal cost (Pascal, Turing, Java).

What Is a Programming Language for?

- Way of thinking: way of expressing algorithms.
- Languages from the user's point of view.
- Abstraction of virtual machine: way of specifying what you want.
- The hardware to do without getting down into the bits.
- Languages from the implementor's point of view.

Why Study Programming Languages? I

- Help you choose a language:
  - C vs. Modula-3 vs. C++ for systems programming.
  - Fortran vs. APL vs. Ada for numerical computations.
  - Ada vs. Modula-2 for embedded systems.
  - Common Lisp vs. Scheme vs. ML for symbolic data manipulation.
  - Java vs. C/CORBA for networked PC programs.
- Make it easier to learn new languages some languages are similar; easy to walk down family tree:
  - Concepts have even more similarity; if you think in terms of iteration, recursion, abstraction (for example), you will find it easier to assimilate the syntax and semantic details of a new language than if you try to pick it up in a vacuum. Think of an analogy to human languages: good grasp of grammar makes it easier to pick up new languages (at least Indo-European).

Why Study Programming Languages? II

- Help you make better use of whatever language you use:
  - Understand obscure features:
    - In C, help you understand unions, arrays & pointers, separate compilation, varargs, catch and throw.
    - In Common Lisp, help you understand first-class functions/closures, streams, catch and throw, symbol internals.
  - Understand implementation costs: choose between alternative ways of doing things, based on knowledge of what will be done underneath:
    - Use simple arithmetic equal (use x*x instead of x**2).
    - Use C pointers or Pascal "with" statement to factor address calculations.
    - Avoid call by value with large data items in Pascal.
    - Avoid the use of call by name in Algol 60.
    - Choose between computation and table lookup (e.g. for cardinality operator in C or C++).

Why Study Programming Languages? III

- Help you make better use of whatever language you use:
  - Figure out how to do things in languages that don't support them explicitly:
    - Lack of suitable control structures in Fortran.
    - Use comments and programmer discipline for control structures.
    - Lack of recursion in Fortran, CSP, etc.
    - Write a recursive algorithm then use mechanical recursion elimination (even for things that aren't quite tail recursive).
  - Figure out how to do things in languages that don't support them explicitly:
    - Lack of named constants and enumerations in Fortran.
    - Use variables that are initialized once, then never changed.
    - Lack of modules in C and Pascal use comments and programmer discipline.
    - Lack of iterators in just about everything fake them with (member?) functions.

Language Groups

- Imperative:
  - von Neumann (Fortran, Pascal, Basic, C).
  - Object-oriented (Smalltalk, Eiffel, C++?).
  - Scripting languages (Perl, Python, JavaScript, PHP).
- Declarative:
  - Functional (Scheme, ML, pure Lisp, FP).
  - Logic, constraint-based (Prolog, VisiCalc, RPG).

- Imperative languages, particularly the von Neumann languages, predominate:
  - They will occupy the bulk of our attention.
- We also plan to spend a lot of time on functional, logic languages.

ANTLR

What is ANTLR?

- ANTLR (http://www.antlr.org/) is a parser generator used to implement language interpreters, compilers, etc.
  - Books by Terence Parr:
    - "The Definitive ANTLR Reference: Building Domain-Specific Languages" http://www.pragprog.com/titles/tpantlr/
    - "Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages" http://www.pragprog.com/titles/tpdsl/
- Most often used to build translators and interpreters for domain-specific languages (DSLs).
- DSLs are usually very high-level languages used for specific tasks and particularly effective in a specific domain.
- DSLs provide a more natural, high-fidelity, robust, and maintainable means of encoding a problem compared to a general-purpose language.

Definitions

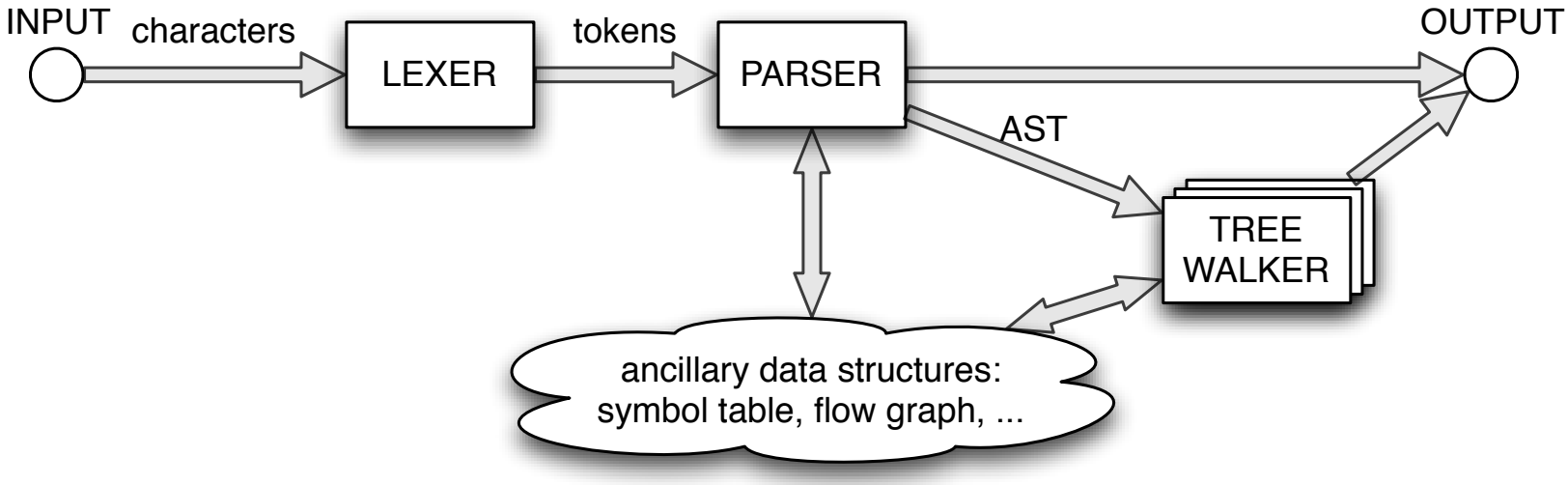
- Lexer: converts a stream of characters to a stream of tokens (ANTLR token objects know their start/stop character stream index, line number, index within the line, and more).
- Parser: processes a stream of tokens, possibly creating an AST.
- Abstract Syntax Tree (AST): an intermediate tree representation of the parsed input that is simpler to process than the stream of tokens and can be efficiently processed multiple times.
- Tree Parser: processes an AST.
- StringTemplate: a library that supports using templates with placeholders for outputting text (ex. Java source code).

Using ANTLR

- Write the grammar using one or more files. A common approach is to use three grammar files, each focusing on a specific aspect of the processing:
  - The first is the lexer grammar: creates tokens from text input.
  - The second is the parser grammar: creates an AST from tokens.
  - The third is the tree parser grammar, which processes an AST.
- This results in three relatively simple grammar files as opposed to one complex grammar file.
- Optionally write StringTemplate templates for producing output.
- Debug the grammar using ANTLRWorks.
- Generate classes from the grammar. These validate that text input conforms to the grammar and execute target language “actions” specified in the grammar.
- Write an application that uses the the generated classes.

Overall Translation Data Flow

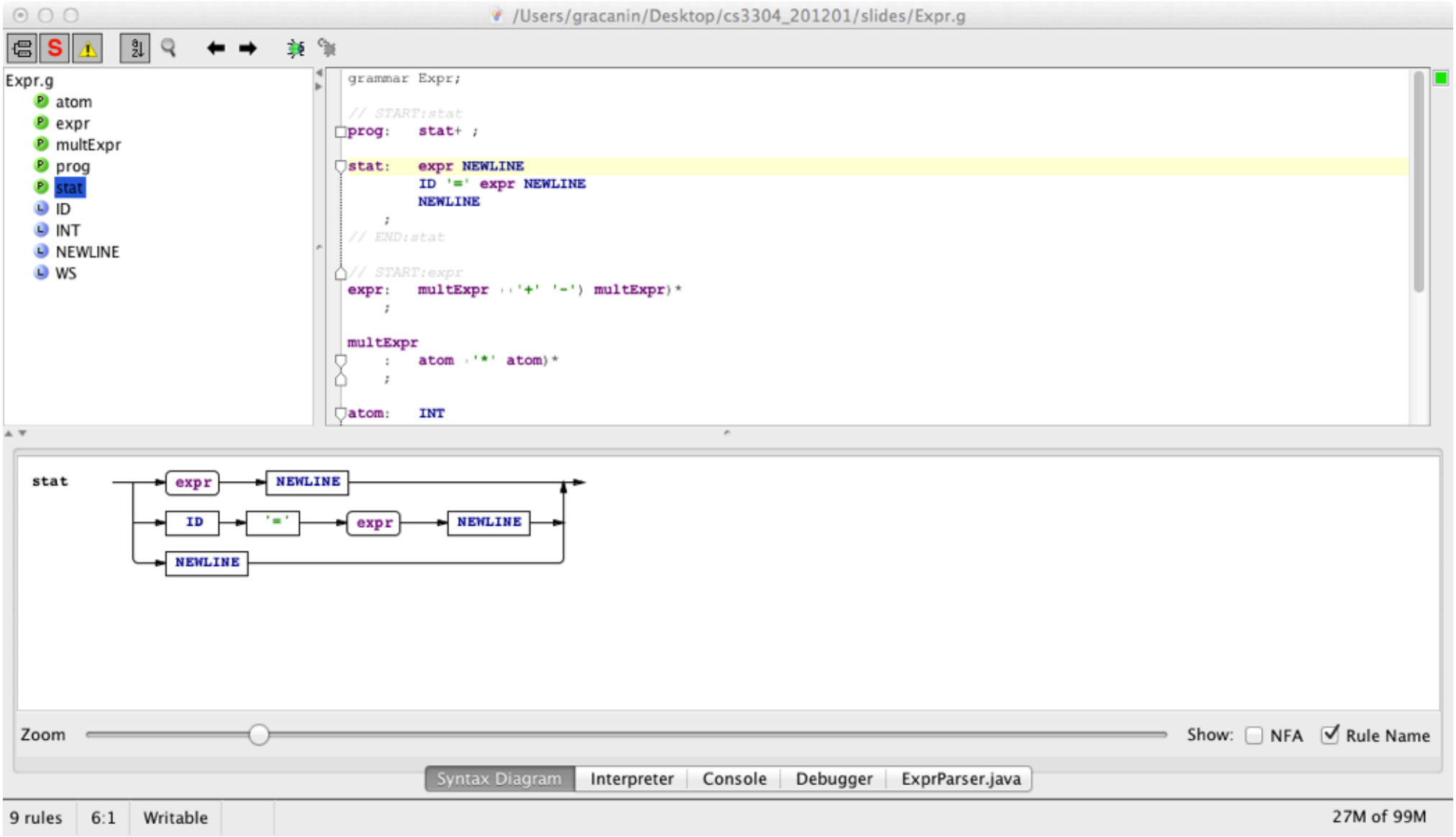
- A translator maps each input sentence to an output sentence.
- The overall translation problem consists of smaller problems mapped to well-defined translation phases (lexing, parsing, and tree parsing).
- The communication between phases uses well-defined data types and structures (characters, tokens, trees, and ancillary structures).
- Often the translation requires multiple passes so an intermediate form is needed to pass the input between phases.
- Abstract Syntax Tree (AST) is a highly processed, condensed version of the input.



Example ANTRL File: `Expr.g`

```
grammar Expr;

prog:   stat+ ;

stat:   expr NEWLINE
    |   ID '=' expr NEWLINE
    |   NEWLINE
    ;

expr:   multExpr (('+'|'-') multExpr)*
    ;

multExpr
    :   atom ('*' atom)*
    ;

atom:   INT
    |   ID
    |   '(' expr ')'
    ;

ID  :   ('a'..'z'|'A'..'Z')+ ;
INT :   '0'..'9'+ ;
NEWLINE:'\r'? '\n' ;
WS  :   (' '|'\t')+ {skip();} ;
```

ANTLRWorks


/Users/gracanin/Desktop/cs3304_201201/slides/Expr.g
Expr.g
atom
expr
multExpr
prog
stat
ID
INT
NEWLINE
WS
grammar Expr;
// START:stat
prog: stat+ ;
stat: expr NEWLINE
ID '=' expr NEWLINE
NEWLINE
;
// END:stat
// START:expr
expr: multExpr (('+' '-') multExpr)*
;
multExpr
: atom ('*' atom)*
;
atom: INT
stat
expr
NEWLINE
ID
'='
expr
NEWLINE
NEWLINE
Zoom
Show:
NFA
Rule Name
Syntax Diagram
Interpreter
Console
Debugger
ExprParser.java
9 rules
6:1
Writable
27M of 99M

Summary

- There are many programming languages.
- Two main groups of programming languages are imperative and declarative.
- ANTLR is a free, open source parser generator tool.