

# cs4240 Compilers, Interpreters and Program Analyzers

[http://www.cc.gatech.edu/classes/AY2005/cs4240\\_fall](http://www.cc.gatech.edu/classes/AY2005/cs4240_fall)

- Ada Gavrilovska - [ada@cc](mailto:ada@cc), 226C
- TA: Guru Prasad Venkatarmani – [guru@cc](mailto:guru@cc)
- textbook:
  - Modern Compiler Implementation in ML  
by Andrew W. Appel
  - ML for the Working Programmer, 2nd Edition  
by Lawrence. C. Paulson
- grading:
  - Midterm (25%), Final (30%)
  - Projects (45%), late policy – 10%/day for 3 days (calendar day)

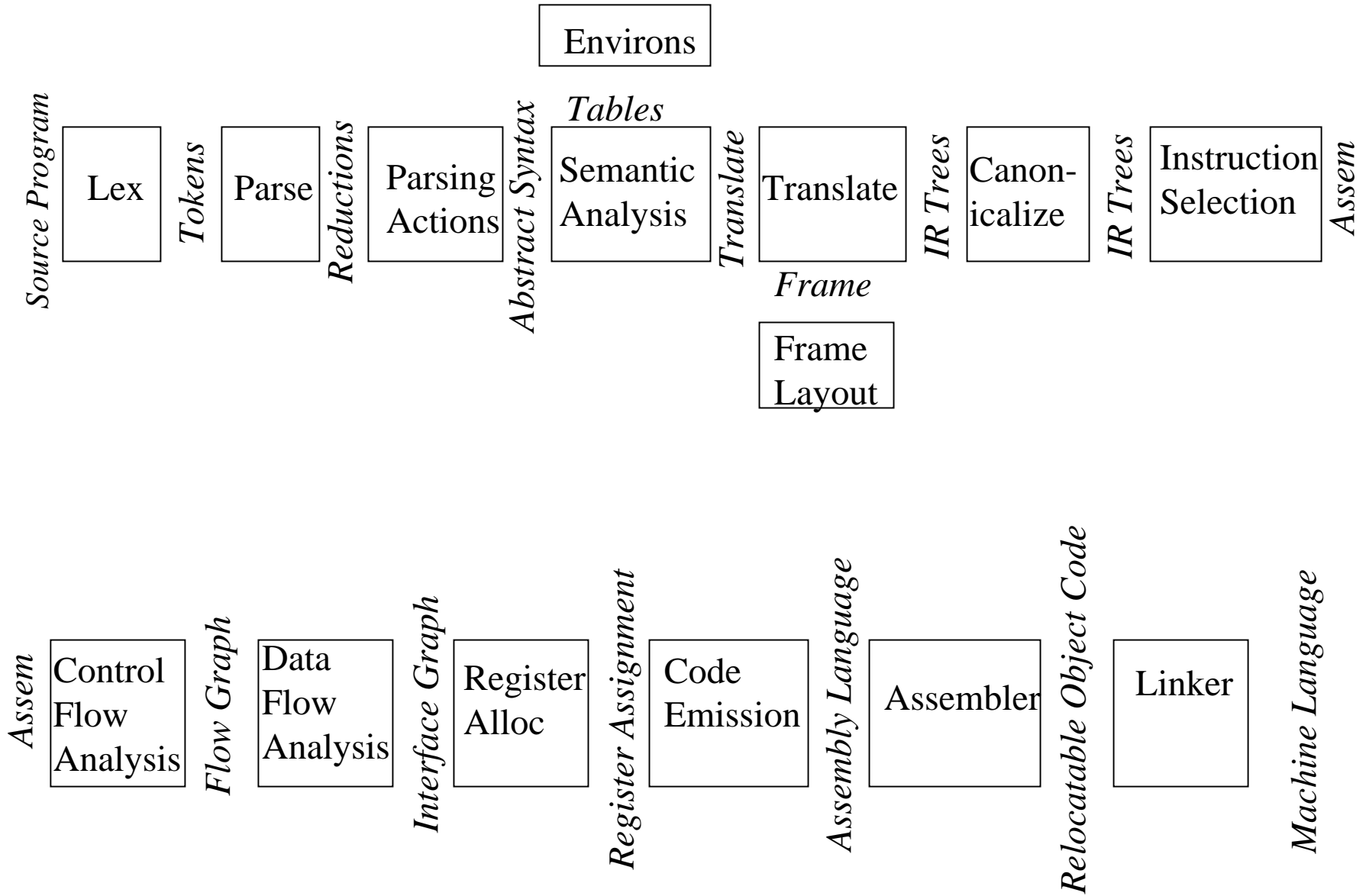
# Class syllabus

- Part I of Appel Textbook
- Select topics from Part II
- Series of Programming Assignments
  - each assignment different phases of a compiler
  - final result compiler for the
  - assignments roughly match Chapters in Part I

# Tiger and ML

- Tiger: simple, yet non-trivial language of Algol family
  - nested functions, record values with implicit pointers, arrays, integer and string variables, few simple structured control loops
- Interfaces between each module - you will be implementing in ML – a strict, statically typed functional programming language
  - (*Standard ML of New Jersey*)
  - <http://www.cs.princeton.edu/~appel/modern/ml>

# Phases of a Compiler and Interfaces between them



- abstractions
  - context-free grammar (for parsing)
    - **context-free grammar:** a grammar in which the left-hand side of each production consists of a single nonterminal symbol.
  - regular expressions (for lexical analysis)
    - **regular expression:** an algebraic expression that denotes a regular language – recognizable by a finite automaton, *e.g.* a simple item such as a variable name or a number in a programming language.
- during compilation – intermediate representations of the program which often take form of a tree with multiple nodes
  - can be described with grammars
  - tree representations more easy to manipulate than source code

# Example with a straight-line program

(straight-line: no loops or if-statements, only expressions and statement)

$a := 5 + 3; b := (\text{print}(a, a - 1), 10 * a); \text{print}( b )$

prints: 8 7

80

---

## 1.3. DATA STRUCTURES FOR TREE LANGUAGES

---

### grammar rules    constructor names

$Stm \rightarrow Stm ; Stm$     (CompoundStm)

$Stm \rightarrow id := Exp$     (AssignStm)

$Stm \rightarrow \text{print} ( ExpList )$     (PrintStm)

$Exp \rightarrow id$     (IdExp)

$Exp \rightarrow \text{num}$     (NumExp)

$Exp \rightarrow Exp Binop Exp$     (OpExp)

$Exp \rightarrow ( Stm , Exp )$     (EseqExp)

$ExpList \rightarrow Exp , ExpList$  (PairExpList)

$ExpList \rightarrow Exp$     (LastExpList)

$Binop \rightarrow +$     (Plus)

$Binop \rightarrow -$     (Minus)

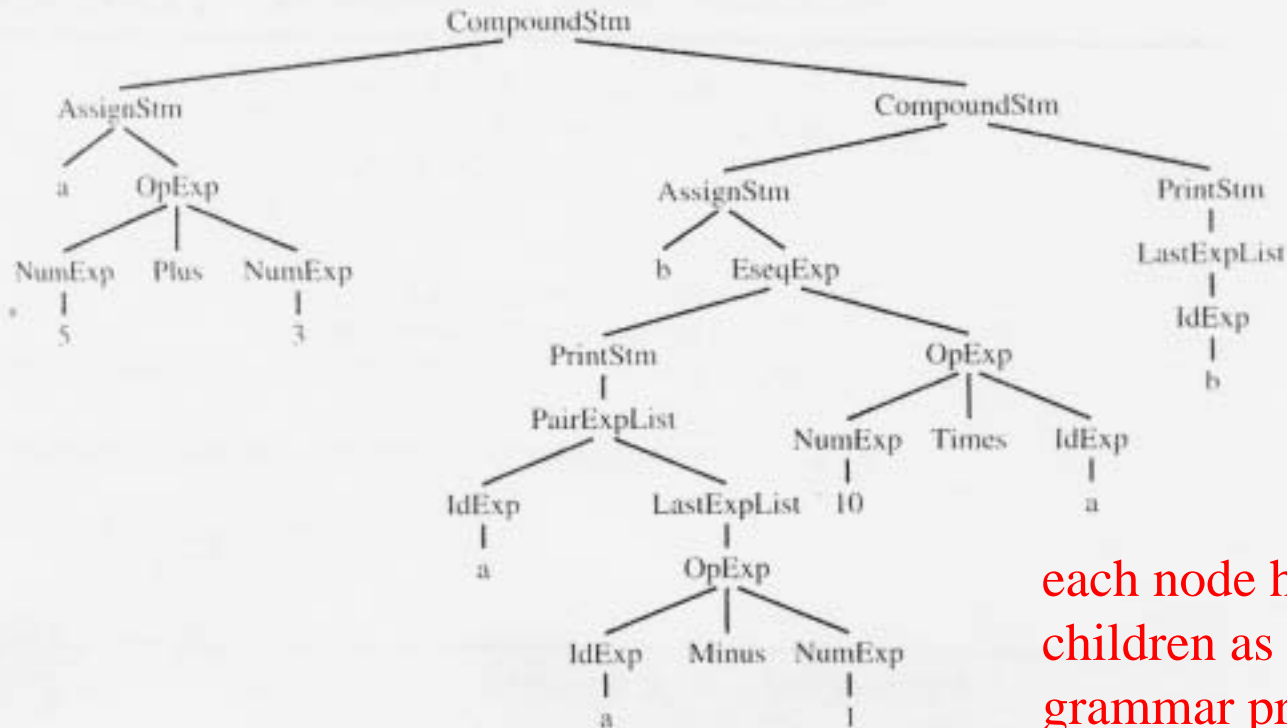
$Binop \rightarrow \times$     (Times)

$Binop \rightarrow /$     (Div)

---

**GRAMMAR 1.3.** A straight-line programming language.

### 1.3. DATA STRUCTURES FOR TREE LANGUAGES



each node has as many children as the corresponding grammar production has right-hand-side symbols

$a := 5 + 3; b := (\text{print}(a, a - 1), 10 * a); \text{print}( b )$

type id=string

datatype binop = Plus | Minus | Times | Div

*Grammar* type

*Stm* stm datatype stm = CompoundStm of stm \* stm

*Exp* exp | AssignStm of id \* exp

*ExpList* exp list | PrintStm of exp list

*id* id and exp = IdExp of id

*num* int | NumExp of Int

| OpExp of exp \* binop \* exp

| Eseq of stm \* exp

```
val prog =  
CompoundStm(  
  AssignStm("a", OpExp(  
    NumExp 5, Plus, NumExp 3)),  
  CompoundStm(  
    AssignStm("b", EseqExp(  
      PrintStm[IdExp "a",  
        OpExp(IdExp "a", Minus, NumExp 1)],  
      OpExp(NumExp 10, Times, IdExp "a"))),  
    PrintStm[IdExp "b"])))
```



# Vocabulary

**lexical:** **1.** refers to information associated with words or symbols in a dictionary or symbol table. **2.** refers to information that can be determined by static examination of a program, *i.e.*, at compile time, without running the program.

**lexical analysis:** parsing and conversion to internal form of the simplest elements of a language, usually specified by a regular grammar, such as variable names, numbers, etc.

**parser:** a program that determines how a given statement in a language could be derived from the grammar of the language, producing a parse tree or other information about the statement as output.

**parser generator:** a program that constructs a parser from a specification of the grammar of a language and actions that are to be taken when phrases of the language are recognized.

**parse tree:** a data structure that shows how a statement in a language is derived from the context-free grammar of the language; it may be annotated with additional information, *e.g.* for compilation purposes.

**parsing:** the process of reading a source language, determining its structure, and producing intermediate code for it.

**semantics:** the meaning of a statement in a language. *cf.* syntax.

**activation record:** stack frame.

**regular expression:** an algebraic expression that denotes a regular language.

**regular grammar:** a grammar that denotes a regular language; its productions can only have on the right-hand side either a terminal string or a terminal string followed by a single nonterminal.

**regular language:** a language described by a regular grammar, or recognizable by a finite automaton, *e.g.* a simple item such as a variable name or a number in a programming language.

**intermediate language:** an internal language used as the representation of a program during compilation, such as trees or quadruples. The source language is translated to intermediate language, then to the object language.

**canonical form:** a standardized form of expressions or data. If all programs put their expressions into a canonical form, the number of cases that will have to be considered by other programs is reduced.

**data flow analysis:** analysis of places in the programs where data receive values and the places that those data values can subsequently be used.

**control flow analysis:** analysis of the possible paths that control flow may take in a program during execution.