

# *Practicum in Database Systems*

*CS4321/CS5321*

Instructor: Alan Demers

Gates 450

[ajd28@cornell.edu](mailto:ajd28@cornell.edu)



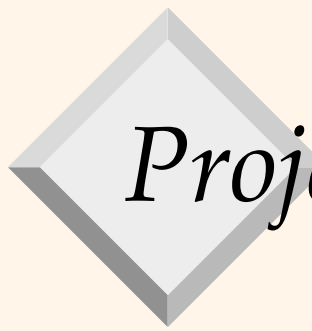
# *Preliminaries*

- Make sure you know how to find
  - CMS
  - Piazza
- Office hours start next week



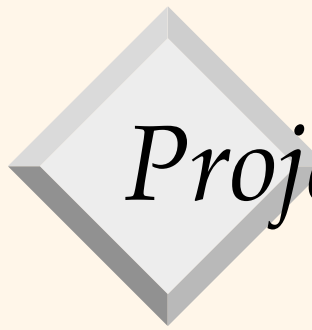
# *What is the course about?*

- Work in groups of 2-3 to develop a simple SQL interpreter
- Process SELECT-FROM-WHERE queries
  - A variety of algorithms for operators (joins etc)
  - Query optimization
- Focus mostly on relational algebra rather than SQL
  - SQL only really a factor in Project 2




# *Projects*

- Project 1: warm-up
- Projects 2-5: develop your interpreter
  - Start from scratch (empty directory)
  - Each project builds on your previous one
    - **No solution code**
  - Our reference implementation is ~5000 LOC




# *Projects*

- Project 1: Warm-up
- Project 2: SQL interpreter on small data
  - in-memory evaluation
- Project 3: Scaling to bigger files
- Project 4: Indexing
- Project 5: Query optimization




# *What you should know*

- Mastery of Java and data structures at CS 2110 level
  - Afraid of tree traversals or recursion? This is not the class for you.
- Projects 1 and 2 will help you to determine if this class is a good fit for you



# *What you should know*

- How to work independently and how to debug your code
  - We provide extensive instructions but you need to read them and expect to spend a lot of time on this
  - 10 minute debugging policy
  - Project 2: will be starting from empty directory
  - Projects 3-5: build on your own code from previous projects – no solution code, no "bail-outs"



# *Grading and logistics*

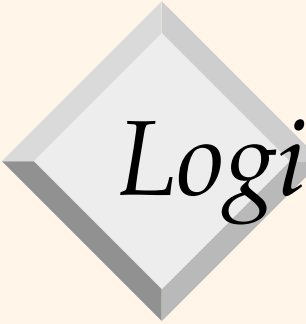
- Projects have various weights, see CMS
- No exams
- For all other logistics, see Course Policies document
  - Must pass a CMS quiz on these to pass course
  - Even if you are taking 4320 this semester also





# *Logistics: due dates and meetings*

- Projects due dates in CMS
- Every project has an initial meeting where I give a brief overview of project
  - Look at instructions before then and bring your questions
  - See schedule in CMS
  - All meetings optional



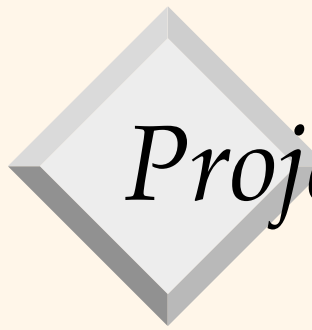
## *Logistics: Eclipse*

- Projects will require you to submit an Eclipse project (and, starting with P2, a runnable command-line .jar)
- You can develop on whatever platform you like, but must submit Eclipse project for grading




# *Project 1*

- No DB material yet
- Instead of manipulating SQL queries, will manipulate simple arithmetic expressions
- Goals:
  - Warm-up
  - Teach or remind you about the basic structure of an interpreter, the visitor pattern, and other key abstractions.



# *Project 1*

- Front-end and back-end
- Intermediate representation: a tree
- Front-end (parser): build tree from string
- Back-end: do stuff to tree
  - convert back to string
  - evaluate to a number
  - convert to list representing same expression in prefix/postfix notation
  - evaluate expression in prefix/postfix notation to a number



# *Prefix and postfix notation*

- Infix notation:  $2 + 3$
- Prefix (Polish) notation:  $+ 2 3$
- Postfix (Reverse Polish) notation:  $2 3 +$
- I expect that you have either seen these or are not particularly freaked out by them
  - Not the case? Wikipedia, Google, StackOverflow etc are your friends



## *Skeleton code*

- Intermediate representations: tree and list
- One class that converts tree to a string
- Uses **visitor pattern**
  - If haven't seen this before, see handout in CMS
  - Will be essential in later projects



# *What you need to implement*

- Recommend to start with back-end
  - easier (probably)
  - familiarizes you with the two intermediate representations
- Back-end: more classes to do stuff to your tree
  - evaluate to a number
  - convert to prefix/postfix
  - print and evaluate from prefix/postfix form



# *What you need to implement*

- Front-end: parser
- Traverse string and build expression tree
- You will build a recursive descent parser
  - If instructions confuse you, see following book for more info: "Compilers: Principles, Techniques and Tools" on reserve at Uris Library





# *Parsers and grammars*

- Parser works based on a grammar that captures the structure of every valid expression

$$E := T \{ + | - T \}^*$$

$$T := F \{ * | / F \}^*$$

$$F := ( E ) \mid - F \mid \text{number}$$

- Parse trees vs expression trees



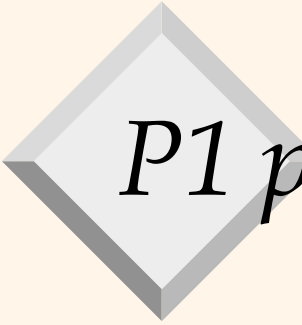
## *Parser skeleton code*

- Three mutually recursive methods, one per nonterminal




# *What you need to implement*

- JUnit tests for all your components
- Must be submitted for grading (and your code must pass your own submitted unit tests)
- We expect you to know what JUnit is and how to use it



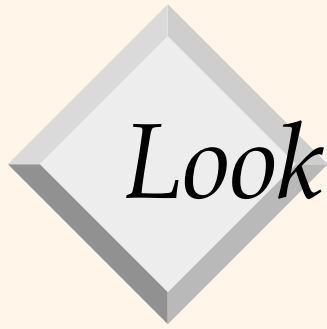
# *P1 peculiarities*

- We know that P1 is "overengineered" for the functionality it provides
  - We know you know how to implement a calculator, the goal is to practice crucial abstractions on something you can't mess up
- P1 is much much easier than any other 4321 project
  - See P2 (posted soon) for comparison
  - Struggling with P1? Consider dropping course.



# *Grading*

- Every Project is graded mostly on automated tests
  - See Course Policies for relevant info on these
- Also some points for code style/comments
  - See instructions
- And some must-have requirements
  - If you violate these, we can impose an arbitrary deduction
  - Deduction could exceed number of "code style" points



## *Looking ahead*

- P2 will be posted in CMS soon if you're curious
  - Note it is much harder/longer
  - Intro meeting Sep 14<sup>th</sup>