COMP 442/6421 Compiler Design

Instructor: Dr. Joey Paquet paquet@cse.concordia.ca
Created By: Zachary Lapointe zachary.lapointe@mail.Concordia.ca
TAs : Dhaval Patel p_haval@encs.concordia.ca
Hamed Jafarpour hamed.jafarpour@concordia.ca

LAB 1 – INTRODUCTION

Why Compiler Design?

- Compilers/Interpreters are a fundamental tool in programming
  - Making and customizing your own tools will make you a coding wizard
  - Gain valuable insight into how compilers work and what limitations they have
    - What can compilers do, and what *can't* they do?
  - Create simple yet powerful *Domain Specific Languages* to express ideas in a programming style, in non programming domains

Why Compiler Design? - DSL

DSL: *Domain Specific Language*

- A specialized *"programming"* language which allows writing *"applications"* for a specific domain
- Contrast with General Purpose Language (C, C++, C#, Java, Python, etc.)
- Can be similar to a programming language, for a specific platform:
  - HTML
  - Unix Shell Scripts
  - SQL
- Or designed for a specific task:
  - Mathematics: *Maple*, *Wolfram, R*
  - Document editing: *LaTeX*, *Markdown, Emacs Lisp*
  - Software building: *Gradle, CMake, Maven*
  - Static analysis tools: Linters, Style checkers, Bug finders, etc.
  - Compute device programming: *GLSL*, *OpenCL C*
  - Music: *Csound*, *Sonic Pi*
  - Anything really: game level design, animation, drawing, chemistry, accounting, you name it!

What the project entails

- Compiler
  - 4 assignments
    - Lexical Analysis
    - Syntax Analysis
    - Semantic Analysis
    - Code Generation
  - The project
    - The cumulative result of the 4 assignments
- If you fall behind, catching up will be difficult
  - How do you stay on schedule?
    - Start early, aim for consistent momentum
    - And . . .

Project - Language Choice

You can use any language

- Pick a language you're familiar with
  - Now is not the time to learn a new language
- Java is supported in the labs and by the TAs

Recommendation: Pick a language where the following is easy . . .

Project - Testing

Manual testing

- Consistency in tests, inputs and results
  - Test files

Automatic testing

- Important for validating your compiler
- Compilers are straightforward to test, since they are stateless at a high level
  - Given an input, they produce an output (source -> tokens)
- Test cases can be made easily from the assignment specifications
- Test often!
- Your tests should be *easy* and *fast* to run

Project - Testing: Junit example

```
//Test if the lexer can separate a string into 2 valid identifiers
@Test
public void identifierTokens_lexed_makes2Tokens() {
      Lexer l = stringLexerFactory("the token");
      Token t1 = l.next();
      Token t2 = l.next();

      assertEquals("'id'",  t1.type);
      assertEquals("the",   t1.lexeme);

      assertEquals("'id'",  t2.type);
      assertEquals("token", t2.lexeme);
}
```

Project - Version Control and Backups

Version control

- Important for any software project
- Very important for a complex software project which are prone to errors, i.e. compilers
- If you haven't done so before, now is a good time to start using version control
  - *SCS Concordia* frequently offers tutorials on the version control system ***Git***

Backups

- **Please, please, please backup your assignments**
  - If using version control, repository systems (GitHub, BitBucket)
    - Free for students
  - Dropbox, OneDrive, email, external hard drive, USB stick
    - ***ANY*** backup is better than none
- Make sure your backups are private, and accessible only to you
  - Not doing so constitutes an **academic offense** under the *Academic Code of Conduct*
  - Private repositories

Theoretical Computer Science - recommended review

The following topics are the foundation to this class, and compiler design in general. Reviewing them is recommended

- Regular Languages
  - Finite State Automata
  - Regular Expression
  - Conversion between the two
- Context Free Grammars
  - Derivation process
  - Push-down automata

AtoCC

A handy, free program, for manipulating *Regular Languages* and *Context Free Grammars*

- Can simulate *automata*

Warning: avoid relying on it too much

- It won't be available during the exam
- It's quite an old program, and has trouble with larger simulations
  - It's prone to crashes
  - e.g. The DFA required for A1

It's best used to check the validity of your transformed regex and grammar

- Live Demo!

Acknowledgement : Zachary Lapointe for creating the presentation