EECS483 D1: Project 1 Overview

Chun-Hung Hsiao

Jan 11, 2013

Special thanks to Ashutosh

Course Websites

- http://www.eecs.umich.edu/courses/eecs483/
  - Schedule, lecture slides
- https://piazza.com/class#winter2013/eecs483001w13
  - Piazza forum for discussions and teammate lookup
- CTools
  - Project & homework materials and grades
  - NO homework/project submission on CTools!

Office Hours

- Prof. Satish Narayanasamy
  - MoWe 3-4p @ 4721 BBB
- Chun-Hung Hsiao
  - TuTh 3-5p @ NE CAEN cluster, 3F Duderstadt Center
- Supriya Rao
  - Fr 3-5p @ NE CAEN cluster, 3F Duderstadt Center

Discussion Sessions

- We'll add a new discussion session starting on next week
- Supriya Rao
  - TBD on Thursdays
- Chun-Hung Hsiao
  - Fr 1:30-2:30p @ 1200 EECS
- The two sessions in the same week will cover the same materials
- Feel free to choose any one of them!

Announcements

- IMPORTANT: Form a group and mail me by Jan 18
  - Email: `chhsiao _AT_ umich _DOT_ edu`
  - Add "`[EECS483]`" at the beginning of the subject
- Project 1 will be announced on Jan 14
- Sign up on Piazza if you haven't done so

Introduction to Projects

- #projects in this course = 5
  - Cover 5 parts of constructing a compiler
- Source language: Decaf
  - A smaller subset of features from Java/C++
- Target language: MIPS
  - Review it if you forget what you’ve learned in 370!
- Project submission procedure will be explained in the next discussion session

Important Notes

- Honor Code
  - You can interact with other students for discussing course materials, provide each other with limited debugging assistance, and help each other learn development tools. Though you can discuss homeworks with others, **you have to write it up individually. Do not refer to others source code for the projects.** We use sophisticated automated program to correlate projects of different groups and check for plagiarism.
    The Engineering Honor Code obligates you not only to abide by this policy, but also to report any violations that you become aware of. Violations of this policy will be brought to the College of Engineering's Honor Council. For more information on the Honor Code, see Honor Council web page. If you have any doubts about whether a certain level of collaboration is permissible, or any other questions, contact the professor.

- Copyright Infringement
  - Understand the scope of copyright law. **Don't take anything from the Internet, or anywhere else**, because it is almost always copyrighted, by default. Don't confuse copyrights, trademarks, and other forms of "intellectual property." Learn about the public domain laws for your jurisdiction.

Stages of the Projects

- Stages of compilation
  - Lexical analysis: scanner (Project 1)
  - Syntax analysis: parser
  - Semantic analysis
  - Code generation
  - Code optimization
- By the end of this term...
  - You would own a compiler!

About Programming Languages...

- Which languages have you used?
- Any idea about compiling programs written in these languages?
- What do you do when you want to read a program written in a language you are not familiar with?

Elements of a Language

- Alphabet
  - Characters on a standard keyboard (in most cases)
- Tokens
  - Keywords, identifiers (names of variables, functions, and classes, etc.), operators, literals
- Grammar
  - Expressions and statements
- Semantic
  - What is the meaning of a keyword?
  - What does an identifier refers to?

Decaf Language Features (1/3)

- Strongly typed, object oriented
- Similar to Java/C++, with a smaller feature set
- Case sensitive, both types of comments are allowed
- Library functions are available for I/O
- Detailed overview available on the course website
  - http://www.eecs.umich.edu/courses/eecs483/decafOverview.pdf

Decaf Language Features (2/3)

- Scoping
  - Global scope
  - Function scope
  - Pair of braces defines a nested scope
  - Java-like scoping rules are followed
- Data types
  - Bulit-in base types: int, double, boolean, string
  - Special base type for functions: void
  - Array type
  - Named type for objects

Decaf Language Features (3/3)

- Supports classes
  - All variables are private
  - Classes are declared globally
  - "`this`" is optional when accessing a member variable
- Single inheritance with multiple interfaces
  - All methods are "virtual"
- Control structure, operator precedence

Decaf Grammar

- A glimpse on Decaf grammar

  StmtBlock ::= `{` *VariableDecl** *Stmt** `}`

  Stmt ::= *Expr* `;` | *IfStmt* | *WhileStmt* | *ForStmt* | *BreakStmt* | *ReturnStmt* | *PrintStmt* | *StmtBlock*

  IfStmt ::= `if` `(` *Expr* `)` *Stmt* `else` *Stmt*

- Consists of tokens (in Courier font) and syntactic variables (in italic)
- Comprehensive reference on the course website
  - http://www.eecs.umich.edu/courses/eecs483/decafOverview.pdf

What’s Project 1 All About?

- Use lex to create a lexical analyzer for Decaf
- Tow parts: preprocessor + scanner
  - The preprocessor takes out comments and handles #define macros
- You just need to implement the scanner
  - Implementing the preprocessor is optional
- Recognize tokens in the order they appear
- Set attributes for every recognized token
  - The information will be used in future projects

Lexical Analysis (1/2)

- In English:

```
Alice      ate      apples     .
```

‹*Noun*, "`Alice`"› ‹*Verb*, "`ate`"› ‹*Noun*, "`apples`"› ‹*Period*›

- In C++/Java/Decaf:

```
i         =      j         +    1         ;
```

‹*Identifier*, "`i`"› ‹*AssignOp*› ‹*Identifier*, "`j`"› ‹*AddOp*› ‹*Literal*, `int 1`› ‹*Semicolon*›

- Why is lexical analysis on English more difficult?

Lexical Analysis (2/2)

- In a programming language, each symbol is associated with a *pattern*
  - Token: pair of symbol and attribute
  - Lexeme: sequence of characters matching the pattern
- In lex, the pattern of a symbol is specified with a *regular expression*
  - Syntax similar to POSIX Extended Regular Expression
  - Similar but not exactly the same as the mathematical notations in your writing homework!

Regular Expression in Lex

- `a` – match “a” in the string
- `[a-z]` – match any lowercase letter from a to z
- `[0-9a-zA-Z]` – match any alphanumerical letters
- `[^a-z]` – match any character but *not* a to z
- `.` – match any character
- `a*` – match 0 or more times of “a”
- `a+` – match at least once of “a”
- `a?` – match at most once of “a”
- `a{`*n*`}` – match exactly *n* times of “a”
- `a|b` – match either “a” or “b”
- `ab` – match “ab”
- `\` – escape character
- `()` – change precedence

Regex Examples

- Date
  - `[0-1][0-9]\.[0-3][0-9]\.[0-9]{4}`
- May, Mary and Harry.
  - `[MH]ar*y`
  - `(Mar?|Harr)y` if you want to match nothing more
- Email id with only small alphabets @ “domain name” “.com”
- Regular expression for a URL?

Lex

- Tool for generating scanners
- Specify the rules of scanning and corresponding action for every rule
- It automatically generates a scanner that would do the trick!
- The scanner is basically a DFA program written in C
- flex: a lex-compatible variant accessible on each CAEN linux machine

Lex Syntax

- Structure of a .l file:

```
DEFINITIONS
%%
RULES
%%
USERCODE
```

Lex Definitions

- Define names for regular expressions used in the RULE section for convenience
- Examples:

```
DIGIT             [0-9]
CommentStart      "/*"
IDENTIFIER        [a-zA-Z][a-zA-Z0-9]*
```

- Necessary user code can be placed here

```
%{
#include <stdio.h>
%}
```

Lex Rules

- Each line consists of a rule in the following form:

  PATTERN ACTION

- Pattern: a combination of defined names and regex
- Action: a piece of C code describing what to do
- Example

```
{DIGIT}+|0x[0-9A-Fa-f]+  val=strtol(yytext,NULL,0);
```

Lex Example: Line Count

```
%{
#include <stdio.h>
int num_lines = 0, num_chars = 0;
%}

%%

\n     ++num_lines; ++num_chars;
.      ++num_chars;

%%

int main() {
    yylex();
    printf("# of lines = %d, # of chars = %d\n",
           num_lines, num_chars);
    return 0;
}
```

References

- http://flex.sourceforge.net/manual/
- http://dinosaur.compilertools.net/#lex
- http://dinosaur.compilertools.net/#flex
- http://epaperpress.com/lexandyacc
- http://www.cs.rug.nl/~jjan/vb/lextut.pdf

Thank you & all the best :)