CMPE 152: Compiler Design

December 5 Class Meeting

Department of Computer Engineering
San Jose State University

Fall 2017
Instructor: Ron Mak
www.cs.sjsu.edu/~mak

Final Exam

- Wednesday, December 13
  - 2:45-5:00 PM in ENG 403
- It will be similar to the midterm.
  - Covers the entire semester.
  - Emphasis on the second half.

Presentation Schedule

- Today
  - Mak & Cheese

- Thursday, Dec. 7
  - Cold Brew
  - Last Minute
  - No Name 1
  - Compile Nation

Lex and Yacc

- Lex and Yacc
  - "Standard" compiler-compiler for Unix and Linux systems.
- Lex automatically generates a scanner written in C.
  - Flex: free GNU version
- Yacc ("Yet another compiler-compiler") automatically generates a parser written in C.
  - Bison: free GNU version
  - Generates a bottom-up shift-reduce parser.

Example: Simple Interpretive Calculator

- Yacc file (production rules): `calc.y`

We'll need to define the **NUMBER** token.

```
...
%token NUMBER
%left '+' '-' /* left associative, same precedence */
%left '*' '/' /* left associative, higher precedence */

%%

exprlist: /* empty list */
      | exprlist '\n'
      | exprlist expr '\n' {printf("\t%lf\n", $2);}
      ;

expr: NUMBER          {$$ = $1;}
      | expr '+' expr {$$ = $1 + $3;}
      | expr '-' expr {$$ = $1 - $3;}
      | expr '*' expr {$$ = $1 * $3;}
      | expr '/' expr {$$ = $1 / $3;}
      | '(' expr ')'  {$$ = $2;}
      ;

%%
```

```
#include <stdio.h>
#include <ctype.h>

int main(int argc, char *argv[])
{
    progname = argv[0];
    yyparse();
}
```

Example: Simple Calculator, *cont'd*

- Lex file (token definitions): `calc.l`

```
%{
#include "calc.tab.h"
extern lineno;
%}
%option noyywrap

%%

[ \t]                        {;}  /* skip blanks and tabs */
[0-9]+\.?|[0-9]*\.[0-9]+ {sscanf(yytext, "%lf", &yylval); return NUMBER;}
\n                           {lineno++; return '\n';}
.                            {return yytext[0];}  /* everything else */
```

- Commands:

```
yacc -d calc.y
lex calc.l
cc -c *.c
cc -o calc *.o
./calc
```

Course Review

- Lectures and PowerPoint slide sets
- Reading assignments
- Homework assignments
- Compiler project

Course Review

- Good understanding of compiler concepts
  - Front end: parser, scanner, and tokens
  - Intermediate tier: symbol table and parse trees
  - Back end: interpreter and code generator
  - The ANTLR 4 compiler-compiler
- Basic understanding of Pascal

Course Review

- What is the overall architecture of a compiler or an interpreter?
  - What are the source language-independent and -dependent parts?
  - What are the target machine-independent and -dependent parts?
- How can we manage the size and complexity of a compiler or an interpreter during its development?

Course Review

- What are the main characteristics of a top-down recursive-descent parser?
- Of a bottom-up parser?

- What is the basic control flow through an interpreter as a source program is read, translated, and executed?
- Through a compiler for code generation?

Course Review

- How do the various components work with each other?
  - parser ←→ scanner
  - scanner ←→ source program
  - parser ←→ symbol table
  - parser ←→ parse tree
  - executor / code generator ←→ symbol table / parse tree

Course Review

- What information is kept in a symbol table?
  - When is a symbol table created?
  - How is this information structured?
  - How is this information accessed?
- What information is kept in a parse tree?
  - When is a parse tree created?
  - How is this information structured?
  - How is this information accessed?

Course Review

- What is the purpose of the
  - symbol table stack
  - runtime stack
  - runtime display
  - operand stack
  - parse stack

Course Review

- Define or explain
  - syntax and semantics
  - syntax diagrams and BNF
  - syntax error handling
  - runtime error handling
  - type checking

Course Review

- Deterministic finite automaton (DFA)
  - start state
  - accepting state
  - transitions
  - state transition table
  - table-driven DFA scanner

Course Review

- What information is kept in an activation record or stack frame?
  - How is this information initialized?
  - What happens during a procedure or function call?
- How to pass parameters
  - by value
  - by reference
- ... with an interpreter
  vs. with generated object code.

Course Review

- The Java Virtual Machine (JVM) architecture
- Runtime stack
- Stack frame
  - operand stack
  - local variables array
  - program counter

Course Review

- The Jasmin assembly language instructions
  - explicit operands
  - operands on the stack
  - standard and “short cut”
  - type descriptors

Course Review

- Jasmin assembler directives:
  - `.class`
  - `.super`
  - `.limit`
  - `.field`
  - `.var`
  - `.method`
  - `.line`
  - `.end`

Course Review

- Basic concepts of the ANTLR 4 compiler-compiler
- Tokens specification with regular expressions
- Production rules
  - labelled alternates
- Tree node visitors
  - Overriding visit methods.

Course Review

- Code generation and code templates
  - expressions
  - assignment statements
  - conditional statements
  - looping statements
  - arrays and records

Course Review

- Compiling procedures and functions
  - fields and local variables
  - call and return
  - passing parameters

Course Review

- Multipass compilers
  - type checking pass with the visitor pattern
  - optimization pass
  - code generation pass with the visitor pattern

Course Review

- Integrating Jasmin routines with Java routines
  - Pascal runtime library
- Instruction selection
- Instruction scheduling
- Register allocation
  - spilling values
  - live variables

Course Review

- Optimization for performance
  - constant folding
  - constant propagation
  - strength reduction
  - dead code elimination
  - loop unrolling
  - common subexpression elimination