CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
Final Exam

5/10/2021, Monday

Start @ 11:45 AM 5/10/2021

End @ 11:45 PM 5/11/2021

See @82 in Piazza
Lexical analysis (LEX) converts the character stream into a sequence of tokens, described by a regular grammar. The token stream is then subjected to syntactic analysis (BISON) to parse the tree, resulting in a context-free grammar. Attribute grammar is then applied to the parse tree. Regular expressions are handled by FLEX, which generates a lexical analyzer for the C program.

Chomsky hierarchy includes Type 0, Type 1, Type 2, and Type 3 grammars. CONTEXT FREE grammar can be built into a state machine corresponding to the grammar. Bison builds the state machine. The symbol table contains all names/symbols and stores info about the symbol, such as its type, kind of value it represents, etc., e.g., for a variable: its type, its scope, etc.
**Semantic Analysis**

- Parse tree
- IR code generation
- IR code
- Machine-Independent optimization
- IR code
- Code generation
- Assembly code

**Action Grammar**

- Attribute grammar
- Semantic rules
- Semantic predicates

**Finite Control**

**Symbol Table**

- Contains all names/symbols used in program
- Info about those symbols, such as type, variable, ...
- For a variable: its type, its scope... how is this done?
- "One table per scope"
Character stream → LEXER → token stream → PARSER → syntax tree

**FRONT END**

### 3-address code instructions

<table>
<thead>
<tr>
<th>IR instructions</th>
<th>dst</th>
<th>src1</th>
<th>op</th>
<th>src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. (X = Y ) op (Z)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. (X = op ) (Y)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. (X = Y)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. GOTO (L_2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. if (x) GOTO (L_3) / ifFalse (x) GOTO (L_1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. if (x) relOp (y) GOTO (L_1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. param (x) / call (f, x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. (x = a[i]) (a[i] = y)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. (x = &amp; y) (x = &amp; y) (x = y)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**HLL**

\[ + \]
\[\begin{align*}
i + i & = i \times i \div i \% i \leq i & r < r + r & = r \times r \div r < c < b < \\
\text{enum } IR-\text{op} & \leftarrow \end{align*}\]

**if** \(x\) GOTO \(L_2\) \(\text{if } x \text{ GOTO } L_1\) \(\text{GOTO } L_1\)

**int** \(\text{foo}(\text{int } x, \text{real } y)\) \(\ldots 3\)
Front End

3-address code instructions

IR instructions

1. \( X = Y \)  \( op \)  \( z \)
2. \( X = op \ Y \)
3. \( X = y \)
4. \( \text{GOTO} \)
5. if \( x \) \( \text{GOTO} \)/ ifFalse \( x \) \( \text{GOTO} \)
6. if \( x \) relop \( y \) \( \text{GOTO} \)
7. param \( x \) / call \( f, n \)

invocation record

\( x = a[i] \)  \( a[i] = y \)
8. \( x = \& y \)  \( x = * y \)  \( *x = y \)

Sem. Analysis

Syntax tree w/ annotations

IR Code Gen

CLA

IR code array

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

if x goto 17
if x goto 17...
gen:

struct IR * instruction = ... 

A[nextInstr++] = instruction;

struct IR {
  struct STE * dst, src1, src2;
  enum IR_op opj
  int jump-target;
  int instruction-type;
}
Exam format

- Expect about 4 short essay questions (choose from about 6). We will use BlueBooks.

- I expect you to take about 30 minutes per question (2 hours combined).

- I expect you to take about 15 minutes per question (1 hour combined) to proofread/edit your responses.
Possible Exam Questions

- Anything from homeworks,

...and...
Type checking
(Semantic processing)

Explain how type errors are detected. Discuss how type information is gathered, stored and checked. Pick a concrete syntactic construct that can contain a type error, and explain how type checking detects the error.
Intermediate Code Generation

Explain how short-circuit Boolean expressions are translated into intermediate code. Discuss how jump targets can be determined. Illustrate by showing how a concrete Boolean expression involving at least two Boolean operators is translated into intermediate code.
Register Allocation and Assignment

Describe the `getReg(I)` algorithm, answering the questions of what data structures it uses, when and how these structures are updated. What is meant by "spill", when does it occur, and why is it needed? Demonstrate with a concrete example.
Symbol Table Usage

- Describe the structure and use of a symbol table. Explain which phases of the compiler use the table, including what data is written to or read from the table during each phase. Give a concrete code example and the corresponding ST.
Invocation Records

Describe a typical layout for an invocation record, detailing what information is stored in the record. Explain how variable length parameters and variable length local data can be accommodated. Discuss the location and use of the stack and top pointers. Give concrete example.
Function Calls

- Explain how a function call takes place. Include in your discussion mention of the roles of the caller and callee in setting up the invocation record, and how machine state is remembered at the call and restored at return. Explain how recursive calls are handled (do NOT discuss tail-call optimization). Give concrete example.
Optimizations

- Pick an optimization and explain the benefit(s) of having the compiler apply it to code, and sketch how it works for a concrete example.
- Ex:
  - tail-call optimization
  - code motion
  - dead code elimination
Thanks for a great semester!

Have a wonderful summer!

Congrats to everyone graduating!!