COMPILER DESIGN (ECS-306)

Course Structure

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Course Code</th>
<th>Course Name</th>
<th>Credits</th>
<th>Details of Sessional Marks</th>
<th>ESM</th>
<th>Total Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CT</td>
<td>TA</td>
<td>Lab</td>
</tr>
<tr>
<td>1.</td>
<td>ICS-603</td>
<td>Compiler Design</td>
<td>4 (3-1-0)</td>
<td>30</td>
<td>20</td>
<td>-</td>
</tr>
</tbody>
</table>

Prerequisite: Theory of Automata and Formal Languages (ECS-305)

Course Content:

Unit-1:
Introduction to Compiler, Phases and passes, Bootstrapping, Finite automata & regular expressions and their applications to lexical analysis, Implementation of lexical analyzers, lexical-analyzer generator, LEX-compiler. The syntactic specification of Programming languages: Context free grammars, derivation and parse trees, capabilities of CFG, Application of grammars in syntax analysis, ambiguity and BNF notation, YACC.

Unit-2:
Basic Parsing Techniques: Parsers, top down parsing, Shift reduces parsing, operator precedence parsing, predictive parsers. Automatic Construction of efficient Parsers: LR parsers, the canonical Collection of LR(0) items, constructing SLR parsing tables, constructing Canonical LR parsing tables, Constructing LALR parsing tables, using ambiguous grammars, an automatic parser generator, implementation of LR parsing tables, constructing LALR sets of items.

Unit-3:
Syntax-directed Translation: Syntax-directed Translation schemes, Implementation of Syntax directed Translators, Intermediate code, postfix notation, Parse trees & syntax trees, three address code, quadruple & triples, translation of assignment statements, Boolean expressions, statements that alter the flow of control, postfix translation, translation with a top down
parser. More about translation: Array references in arithmetic expressions, procedures call, declarations, Case statements.

Unit-4:


Unit-5:

Introduction to code optimization: Loop optimization, the DAG representation of basic blocks, value numbers and algebraic laws, Global Data-Flow analysis.

Text and References Books:


Course Outcomes:

1. Describe the role of each phase of a compiler with its construction tools. (Understand)
2. Develop a Lexical Analyzer for recognizing tokens of a given language with an understanding of symbol table management and error handling. (Apply)
3. Construct top-down, bottom-up, operator precedence and SLR parsers with an understanding of Context Free Grammars and syntax analysis. (Apply)
4. Design and develop semantic analyzers for type-checking and intermediate code generators to translate the source program into an intermediate code. (Apply)
5. Construct code optimizers to optimize the target code generated. (Apply)

Lesson Plan

LECTURE PLAN

COMPILER DESIGN (ICS-603)

<table>
<thead>
<tr>
<th>Lecture No.</th>
<th>Unit/Topic/Subtopic</th>
<th>Teaching Method(s)</th>
<th>Learning Material</th>
<th>Remarks if any</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIT 1:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>Introduction to Compiler, Phases and passes, Bootstrapping</td>
<td>Chalk Board</td>
<td>Books and Notes</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Finite automata and regular expressions and their applications to</td>
<td>Chalk Board</td>
<td>Books and Notes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>Implementation of lexical analyzers, lexical-analyzer generator, LEX compiler.</td>
<td>Chalk and Board</td>
<td>Books and Notes</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>The syntactic specification of Programming languages: Context free grammars,</td>
<td>Chalk and Board</td>
<td>Books and Notes</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>derivation and parse trees, capabilities of CFG.</td>
<td>Chalk and Board</td>
<td>Books and Notes</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>Application of grammars in syntax analysis.</td>
<td>Chalk and Board</td>
<td>Books and Notes</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>ambiguity and BNF notation, YACC.</td>
<td>Chalk and Board</td>
<td>Books and Notes</td>
<td></td>
</tr>
</tbody>
</table>

**UNIT - 2**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8.</td>
<td>Basic Parsing Techniques: Parsers, top down parsing.</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>9.</td>
<td>Shift reduces parsing</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>10.</td>
<td>operator precedence parsing</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>11.</td>
<td>predictive parsers.</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>12.</td>
<td>Automatic Construction of efficient Parsers: LR parsers,</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>13.</td>
<td>the canonical Collection of LR(0) items</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>14.</td>
<td>the canonical Collection of LR(0) items</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>15.</td>
<td>constructing SLR parsing tables,</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>16.</td>
<td>constructing SLR parsing tables,</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>17.</td>
<td>constructing Canonical LR parsing tables,</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>18.</td>
<td>constructing Canonical LR parsing tables,</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>19.</td>
<td>Constructing LALR parsing tables, using ambiguous grammars,</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>20.</td>
<td>Constructing LALR parsing tables, using ambiguous grammars,</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td>21.</td>
<td>An automatic parser generator,</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td></td>
<td>Implementation of LR parsing tables,</td>
<td>Board and Notes</td>
</tr>
<tr>
<td>---</td>
<td>-------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>22.</td>
<td>constructing LALR sets of items.</td>
<td>Chalk and Board</td>
</tr>
<tr>
<td></td>
<td>UNIT- 3</td>
<td></td>
</tr>
<tr>
<td>23.</td>
<td>Syntax-directed Translation schemes, Implementation of Syntax directed Translators,</td>
<td>Ppt</td>
</tr>
<tr>
<td>24.</td>
<td>Intermediate code, postfix notation,</td>
<td>Ppt</td>
</tr>
<tr>
<td>25.</td>
<td>Parse trees &amp; syntax trees,</td>
<td>Ppt</td>
</tr>
<tr>
<td>26.</td>
<td>Three address code, quadruple &amp; triples,</td>
<td>Ppt</td>
</tr>
<tr>
<td>27.</td>
<td>Translation of assignment statements, Boolean expressions, statements that alter the flow of control,</td>
<td>Ppt</td>
</tr>
<tr>
<td>28.</td>
<td>Postfix translation, translation with a top down parser postfix translation</td>
<td>Ppt</td>
</tr>
<tr>
<td>29.</td>
<td>translation with a top down parser</td>
<td>Ppt</td>
</tr>
<tr>
<td>30.</td>
<td>More about translation: Array references in arithmetic expressions,</td>
<td>Ppt</td>
</tr>
<tr>
<td>31.</td>
<td>Procedures call, declarations, case statements.</td>
<td>Ppt</td>
</tr>
<tr>
<td></td>
<td>UNIT-4 :</td>
<td></td>
</tr>
<tr>
<td>32.</td>
<td>Symbol Tables: Data structure and representing scope information.</td>
<td>Ppt</td>
</tr>
<tr>
<td>33.</td>
<td>Run-Time Administration: Implementation of simple stack allocation scheme</td>
<td>Ppt</td>
</tr>
<tr>
<td>34.</td>
<td>storage allocation in block structured language</td>
<td>Ppt</td>
</tr>
<tr>
<td>35.</td>
<td>Error Detection &amp; Recovery: Lexical Phase errors,</td>
<td>Ppt</td>
</tr>
<tr>
<td>36.</td>
<td>Syntactic phase errors semantic errors.</td>
<td>Ppt</td>
</tr>
<tr>
<td></td>
<td>UNIT-5 :</td>
<td></td>
</tr>
<tr>
<td>37.</td>
<td>Introduction to code optimization.</td>
<td>Ppt</td>
</tr>
<tr>
<td>38.</td>
<td>Loop optimization</td>
<td>Ppt</td>
</tr>
<tr>
<td>39.</td>
<td>The DAG representation of basic blocks</td>
<td>Ppt</td>
</tr>
<tr>
<td>40.</td>
<td>Value numbers and algebraic laws, Global Data-Flow analysis.</td>
<td>Ppt</td>
</tr>
</tbody>
</table>
Assignment 1
Instructor: Ankita Gautam
Compiler Design (2018-19)

- Assignment should be handwritten.
- Don’t Copy
- Deadline: 13/02/2019 (5.00 Pm)

1. Write short note on bootstrapping.
2. Define the terms Language Translator and compiler.
3. Write Regular Expression for specifying Identifiers and constant of C.
4. Define LEX. Explain the use and form of lex program with an example.
5. Explain the need for dividing the compilation process into various phases and explain its functions.
6. Define Regular Expressions and Regular Grammar.
7. How to remove Left Factoring and Left recursion in a grammar.
8. What is ambiguous grammar? Explain with the help of an example.
9. What are the applications of Finite automata and regular expression in Lexical analysis?
10. Calculate first and follow for the following grammar?

   \[
   S \rightarrow \text{xABC} \\
   A \rightarrow \text{a|bbD} \\
   B \rightarrow \text{a|ε} \\
   C \rightarrow \text{b|ε} \\
   D \rightarrow \text{c|ε}
   \]

11. Construct the recursive decent parser for the string id*(id+id) following grammar?

   \[
   E \rightarrow \text{E+T/T} \\
   T \rightarrow \text{T*F/F} \\
   F \rightarrow (E)/id
   \]

12. Consider the grammar

   \[
   S \rightarrow \text{xABC} \\
   A \rightarrow \text{a|bbD} \\
   B \rightarrow \text{a|ε} \\
   C \rightarrow \text{b|ε} \\
   D \rightarrow \text{c|ε}
   \]

   Derive the predictive parsing table.

Assignment 2
Instructor: Ankita Gautam
Compiler Design (2018-19)

- Assignment should be handwritten.
- Don’t Copy
- Deadline: 13/03/2019 (5.00 Pm)
- Give explanation for all the MCQs
1. For a given grammar if SLR(1) has \( n_1 \) states, LALR(1) has \( n_2 \) states, LR(1) has \( n_3 \) states which of the following is true?
   a. \( n_2 = n_3 \)
   b. \( n_2 \leq n_3 \)
   c. \( n_2 < n_3 \)
   d. none

2. Consider the grammar given below:
   
   \[
   \begin{align*}
   &A \rightarrow SB \mid S \\
   &B \rightarrow ; SB \mid ; S \\
   &S \rightarrow a
   \end{align*}
   \]
   Convert it into operator grammar.

3. Consider SLR(1) and LALR(1) tables for CFG. Which of the following is false?
   a. Goto of both tables may be different
   b. Shift entries are identical in both tables
   c. Reduce entries in tables may be different
   d. Error entries in tables may be different

4. Consider the grammar \( S \rightarrow CC, C \rightarrow cC \mid d \) is
   a. LL(1)
   b. SLR(1) but not LL(1)
   c. LALR(1) but not SLR(1)
   d. LR(1) but not LALR(1)

5. Which of the following is the most powerful parsing method?
   a. LL(1)
   b. CLR(1)
   c. SLR(1)
   d. LALR(1)

6. The following grammar is \( \)\[
   \begin{align*}
   &S \rightarrow ABC \\
   &A \rightarrow 0A1 \mid \varepsilon \\
   &B \rightarrow 1B \mid \varepsilon \\
   &C \rightarrow 1C0 \mid \varepsilon \\
   &\text{a. LL(1)} \\
   &\text{b. LR(0)} \\
   &\text{c. not LL(1)} \\
   &\text{d. not LL(1) and not LR(0)}
   \end{align*}
   \]

7. Check whether the following grammar is LR(0), SLR(1).
   \( E \rightarrow bEa \mid aEb \mid ba \)

8. Check if the following grammar is SLR(1), CLR(1).
   \( S \rightarrow L = R \mid R \)
L → * R | id
R → L

9. How many conflicts occur in DFA with LR(1) items for the following grammar?

\[ S \to SS \mid a \mid c \]

10. Consider the following two sets of LR(1) items of an LR(1) grammar.

\[
\begin{align*}
X & \rightarrow c.X, c/d \\
X & \rightarrow .cX, c/d \\
X & \rightarrow .d, c/d \\
X & \rightarrow c.X, $ \\
X & \rightarrow .cX, $ \\
X & \rightarrow .d, $
\end{align*}
\]

1. Cannot be merged since look aheads are different.
2. Can be merged but will result in S-R conflict.
3. Can be merged but will result in R-R conflict.
4. Cannot be merged since goto on c will lead to two different sets.

a. 1 only
b. 2 only
c. 1 and 4 only
d. 1, 2, 3, and 4

Assignment 3

1. Explain syntax directed definition.
2. Explain the Translation scheme of SDD.
3. Draw the syntax tree and DAG for the following expression:

\[(a*b)+(c-d)*(a*b)+b\]
4. Differentiate between synthesized translation and inherited translation.
5. What is intermediate code and write the two benefits of intermediate code generation.
6. Write short notes on:
   a. Abstract syntax tree
   b. Polish notation
7. Write quadruples, triples and indirect triples for the expression:
   \(-(a*b)+(c+d)-(a+b+c+d)\)

**Assignment 4**

1. Define Symbol table. Explain different types of Data structure for symbol table
2. Draw the format of Activation Record in stack allocation and explain each field in it.
3. Distinguish between static scope and dynamic scope. Briefly explain access to non-local names in static scope.
4. What is meant by data flow equation?
5. What is activation record? Write the various fields of Activation Record.
6. What are the functions of error handler?
8. Classify the errors and discuss the errors in each phase of Compiler.

**Assignment 5**

1. Construct the DAG for the following basic blocks
   a. \(t1:=4*i\)
   b. \(t2:=a[t1]\)
   c. \(t3:=4*i\)
   d. \(t4:=b[t3]\)
   e. \(t5:=t2*t4\)
   f. \(t6:=\text{prod}+t5\)
   g. \(\text{prod}:=t6\)
   h. \(t7:=i+1\)
   i. \(i:=t7\)
   j. if \(i<=20\) goto 1

2. Explain the simple code generator and generate target code sequence for the following statement \(d:=(a-b)+(a-c)+(a-c)\)

3. What is the role of peephole optimization in compilation process.
4. Consider the following C code segment
   for \((i = 0, i<n; i++)\)
   {
for (j=0; j<n; j++)
{
    if (i%2)
    {
        x += (4*j + 5*i);
        y += (7 + 4*j);
    }
}

Which one of the following is false?
A: The code contains loop invariant computation
B: There is scope of common sub-expression elimination in this code
C: There is scope of strength reduction in this code
D: There is scope of dead code elimination in this code

5. Consider the following code segment.
   x = u - t;
   y = x * v;
   x = y + w;
   y = t - z;
   y = x * y;
   The minimum number of total variables required to convert the above code segment to static single assignment form is:

6. What is control and data flow analysis? Explain with example.

7. Write a short note on:
   a. Flow graph (with example)
   b. Dominators
   c. Natural loops
   d. Inner loops
   e. Reducible flow graphs