CSE 312 LANGUAGE PROCESSORS
LAB MANUAL

VI Sem, BE (CS&E)

2012

Prepared By

Approved By

(Dr. Ashalatha Nayak)
(Jaya Krishna R)
(Ashwath Rao)  (H.O.D)

DEPT OF COMPUTER SCIENCE & ENGG.
Week-wise Schedule

Week 1 : Preliminary Scanning Applications
Week 2 : Identification of Tokens in a given Program
Week 3 & 4 : Design of Lexical Analyzer
Week 5, 6 & 7 : Design of Parser
Week 8, 9 & 10 : Design of Code Generator
Week 11 : Usage of Lex
Week 12 : Usage of Yacc
Week 13 & 14 : Final Test
PROCEDURE OF EVALUATION

Student will be evaluated based on following criteria:

- Scanning : 5 Marks
- Tokenizing : 5 Marks
- Lexical analyzer : 10 Marks
- Parser : 15 Marks
- Code generation : 15 Marks
- LEX & Yacc : 10 Marks
- Test : 40 Marks (10 Write UP + 5 Viva + 25 Exn.)

Total : 100 Marks
WEEK 1: Preliminary Scanning applications

1) Write a program which will take as input a C program consisting of single and multi-line comments and multiple blank spaces and produces as output the C program without comments and single blank space.

// This is a single line comment

/* *****This is a
****** Multiline Comment
******/

2) Write a program, which will read a program written in C, recognize all of the keywords in that program and print them in upper case letters.

WEEK 2: Identification of Tokens in a given Program

Write a program, which will read a program written in C, recognize all of the lexemes (int, float, char, for, while etc., ids, +, -, /, *, (,), numbers, <,>, <=, >=, !=, ==, <>, <=, >=, !=, ==), tokens (keywords, identifiers, operators, special symbols, relational operators) and display them in a separate file.
WEEK 3 – 10: Design of Mini Compiler for C Language for the given subset

Data Types : int, char
Arrays : 1-dimensional
Expressions : Arithmetic and Relational
Looping statements : for, while
Decision statements : if, if – else

The formal grammar (sample):

Program - main () { declarations statement-list }

declarations → data-type identifier-list; declarations | ε
data-type → int | char
identifier-list → id | id, identifier-list | id[number] , identifier-list | id[number]
statement_list → statement ; statement_list | ε
statement → assign-stat | decision_stat | looping-stat
assign_stat → id = expn
expn → simple-expn eprime
eprime → relop simple-expn | ε
simple-exp → term seprime
seprime → addop term seprime | ε
term → factor tprime
tprime → mulop factor tprime | ε
factor → id | num
decision-stat → if ( expn ) stat dprime
dprime → else stat | ε
looping-stat → while (expn) stat | for (assign_stat ; expn ; assign_stat ) stat
relop → == | != | <= | >= | > | <
addop → + | -
mulop → * | / | %
WEEK 3 & 4: Design of Lexical analyzer

To construct an adhoc Lexical Analyzer.

- Identifying different classes of tokens like: keywords, identifiers and special symbols.
- Selecting a suitable data structure for symbol table (the alternates are linked list, hashing, array of structures, binary search tree)
- Having selected a data structure, identifying the appropriate fields.

To test the Lexical Analyzer:

Input: C Program
Output: Tokens and their Class

Interface:

The Lexical Analyzer should tokenize a given source program and return the next token and it’s class whenever the parser requests.
WEEK 5, 6 AND 7: Design of a Predictive Parser

To code and test parser:

- Students should write a formal grammar for the given C subset (Refer Sample Grammar given above)

- Remove left recursion from each of the productions so that the underlying grammar can be parsed with a predictive parser.

- The parser obtains a string of tokens from the lexical analyzer and verifies that the string can be generated by the grammar for the C language.

- The parser should report syntax errors if any (for eg.: Misspelling an identifier or keyword, Undeclared or Multiply declared identifier, Arithmetic or Relational Expressions with unbalanced parentheses and Expression syntax error etc.) with appropriate line-no.
WEEK 8, 9 and 10: Design of Code generator

- The target code to be generated is 8086 assembly language program.
- Registers have to be selected for each of the variables used by the program.
- Code generator will take each line of the source program and generate its equivalent assembly code.

WEEK 11 AND 12: Usage of LEX and Yacc

WEEK 11:

1. Design a lexical analyzer for recognizing following class of tokens: Keywords (if, then, else), id, num and relational operators using lex compiler.

2. Write a lex program to count the number of printf and scanf statements in a valid C program and replace them with write and read statements respectively.

3. Write a lex program to count the number of characters, words, blanks and lines in a given string.

WEEK 12:

1. Write a yacc program to test the validity of a simple expression involving the operators +,-,*,./.

2. Write a yacc program to recognize nested if statements and display the number of levels of nested if.
WEEK 13: Test

Test Portion: Questions involving concepts of Lexical Analysis, Syntax Analysis and Code Generation

REFERENCES:

1. Principles of Compiler Design- Aho, Sethi and Ullman
2. Principles of Compiler Design- Aho and Ullman
3. Compiler Construction in C – Allen I. Holub
4. Engineering a Compiler-Cooper and Torczon, Morgan-Kaufmann Publishers
5. Compiler Construction, Principles and Practice- Kenneth C. Louden

****