MANIPAL INSTITUTE OF TECHNOLOGY
(Constituent Institute of MANIPAL University)
MANIPAL-576104

CSE 312 LANGUAGE PROCESSORS LAB MANUAL

VI Sem , BE (CS&E)

2014

Prepared By

1. **Dr. Ashalatha Nayak**
2. **Ms. Priya Kamath B.**
3. **Ms. Deepthi S.**
4. **Ms. Ancilla Pinto**

Approved By

(H.O.D)

DEPT OF COMPTER SCIENCE & ENGG.

M. I. T., MANIPAL

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 & 8	:	Design of Parser
Week 9 & 10	:	Design of Code Generator
Week 11 & 12	:	Usage of Lex
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	:	20 Marks
Code generation	:	10 Marks
LEX	:	10 Marks
Test	:	40 Marks (15 write up+ 25 Execution.)
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

Program - main () { declarations statement-list }

declarations→ data-type identifier-list; declarations | ∈

data-type → int | char

identifier-list → id | id, identifier-list | id[number] , identifier-list | id[number]

statement_list → statement ; statement_list | ∈

statement → assign-stat | decision_stat | looping-stat

assign_stat → id = expn

expn→ simple-expn eprime

eprime→relop simple-expn|∈

simple-exp→ term seprime

seprime→addop term seprime |∈

term → factor tprime

tprime → mulop factor tprime |∈

factor → id | num

decision-stat → if (expn) stat dprime

dprime → else stat | ∈

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 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, 7 and 8: Design of a Recursive Descent Parser

To code and test parser:

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

Simple grammars

1. E→num T
   T→*num T | ϵ

2. S→a | (T)
   T→T, S|S

3. E→E+T | T
   T→T*F |F
   F→€ | id

4. S→aAcBe
   A→Ab|b
   B→d

5. Build a RD parser for the C grammar.

WEEK 8 and 9: Design of Code generator

WEEK 11 and 12: Usage of LEX

1. Write a LEX program to count the number of lines, words, blank spaces and characters in a given input.
2. Write a LEX program to find the number of vowels, consonants in the given input.
3. Write a LEX program to check if a given number is positive or negative.
4. Write a LEX program to check if a given statement is simple or compound.
5. Write a LEX program to count the type of numbers.
6. Write a LEX program to check the validity of a given arithmetic statement.
7. 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.
8. Write a LEX program to check if a given number is even or odd.

WEEK 13 and 14: Test

REFERENCES

1. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, “Compilers Principles, Techniques and Tools”, Pearson Education, 2nd edition. 2010
2. Kenneth C. Louden, “Compiler Construction - Principles and Practice”, Thomson, India Edition, 2007.

****