MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NAAC with 'A' Grade & Recognized Under Section 2(f) & 12(B) of the UGC act,1956

COURSE CONTENT

COMPILER DESIGN LABORATORY								
VI Semester: CSE								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
2460584	**Professional Core courses**	0	0	2	1	40	60	100
Contact Classes: 0	**Tutorial Classes: 0**	**Practical Classes: 2**				**Total Classes: 0**		
Prerequisites: **Java Programming Lab, Web Technologies Lab**								

Course Overview:

The Compiler Design Lab provides hands-on experience in building the key components of a compiler. Students learn to use LEX for lexical analysis, implement parsing algorithms like LL(1), SLR(1), and LALR, and generate Three-Address Code (TAC) for arithmetic and control statements. The lab covers designing a mini-compiler for a simple language with variable declarations, expressions, loops, conditionals, and 1-dimensional arrays, while handling scoping rules. Experiments focus on scanning tokens, syntax analysis, parsing, and code generation. By the end of the course, students gain practical skills in compiler construction, syntax tree development, and program evaluation, providing a strong foundation for advanced studies in programming languages and compiler design.

Course Objectives:

1. To apply lexical analysis techniques using LEX for identifying tokens, reserved words, and identifiers of a programming language.
2. To analyze and implement parsing algorithms such as Predictive (LL(1)), SLR(1), and LALR parsers for given grammars.
3. To develop programs for intermediate code generation, producing Three-Address Code (TAC) for arithmetic and control statements.
4. To design and implement bottom-up parsing techniques for constructing syntax trees and handling ambiguous grammars.
5. To evaluate and implement a mini-compiler for a simple programming language, covering variable declarations, expressions, control structures, and scoping rules.

Course Outcomes: After Completion of the Course, Students should be able to

1. Apply lexical analysis techniques using LEX for identifying tokens, reserved words, and identifiers of a programming language.
2. Analyze and implement parsing algorithms such as Predictive (LL(1)), SLR(1), and LALR parsers for given grammars.
3. Apply intermediate code generation techniques by developing programs for producing Three-Address Code (TAC) for arithmetic and control statements.
4. Analyze complex grammars and implement bottom-up parsing (SLR/LALR) for constructing syntax trees and handle ambiguity.
5. Evaluate the design and implementation of a mini-compiler for a simple programming language with variable declarations, expressions, control structures, and scoping rules.

LIST OF EXPERIMENTS

Compiler Design Experiments

1. Write a LEX Program to scan reserved word & Identifiers of C Language
2. Implement Predictive Parsing algorithm
3. Write a C program to generate three address code.
4. Implement SLR (1) Parsing algorithm
5. Design LALR bottom-up parser for the given language

< program>::=<block>

<block>::={< variable definition >< slist >}

|{<slist>}

<variable definition>::=int<vardef list>;

<vardezflist>::=<vardec>|<vardec>,<vardeflist>

<vardec>::=<identifier>|<identifier>[<constant>]

<slist>::=<statement>|<statement>;<slist>

<statement>::=<assignment>|<ifstatement>|<whilestatement>

|<block>|<printstatement>|<empty>

<assignment>::=<identifier>=<expression>

|<identifier>[<expression>]=<expression>

<ifstatement> ::=if<bexpression> then<slist> else<slist>endif

|if<bexpression>then<slist>endif

<whilestatement>::=while<bexpression>do<slist

>enddo

<printstatement>::=print(<expression>)

<expression>::=<expression><addingop><term>|<term>|<addingop><term>

<bexpression>::=<expression><relop><expression>

<relop>::= <|<= |==|>=|>|!=

<addingop>::=+|-

<term>::=<term><multop><factor>|<factor>

<multop>::=* |/

<factor>::=<constant>|<identifier>|<identifier>[<expression>]

| (<expression>)

<constant>::=<digit>|<digit><constant>

<identifier>::=<identifier><letterordigit>|<letter>

<letterordigit>::=<letter>|<digit>

<letter>::=a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<digit>::=0|1|2|3|4|5|6|7|8|9

<empty>hasthe obvious meaning

Comments (zero or more characters enclosed between the standard C/Java-style comment brackets /*...*/) can be inserted. The language has rudimentary support for 1- dimensional arrays. The declaration int a[3] declares an array of three elements, referenced as a[0], a[1] and a[2].

Note al so that you should worry about the scoping of names. A simple program written in this language is:

```
{
    int a[3], t1, t2; t1= 2;
    a[0] =1;
    a[1] =2;
    a[t1] =3;
    t2=-( a[2] + t1 * 6 ) / ( a[2] - t1 );
    if   t2 > 5 then print(t2); else
    {
        int      t3;
            t3=99;
            t2=-25;
        print(-t1 +t2 * t3 ) ; /*thisisa comment on2lines*/
    }
    endif
}
```

TEXT BOOKS:

1. Compilers: Principles, Techniques and Tools, Second Edition, Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffry D. Ullman.

REFERENCE BOOKS:

1. Lex&Yacc – John R. Levine, Tony Mason, Doug Brown, O'reilly Compiler Construction by Louden, Thomson

ELECTRONIC RESOURCES:

1 https://www.geeksforgeeks.org/compiler-design-tutorials/

2 https://www.geeksforgeeks.org/compiler-design/compiler-design-for-gate/

3 https://www.cerebrixacademy.com/compiler-design-tutorial/

4 https://cse.iitpkd.ac.in/courses/cs3140-Compiler-Design-Laboratory/

5 https://www.tutorialspoint.com/course/msc-semester-iii-compiler-design/index.asp

MATERIALS ONLINE:

1. Course template
2. Open-ended experiments
3. Definitions and terminology
4. Lab Manual