IIT
HYDERABAD

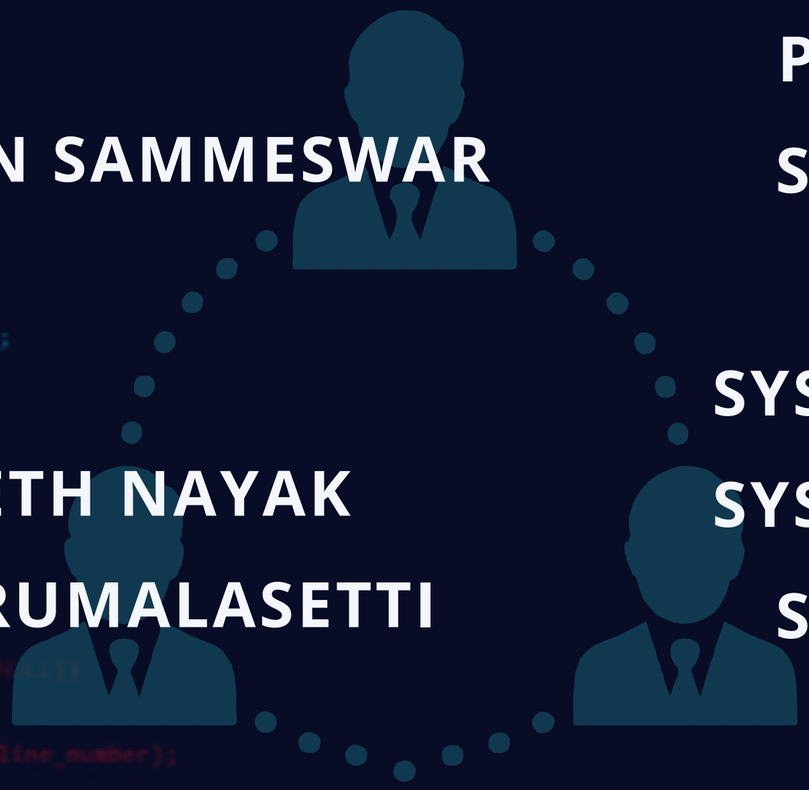
LiTeC

FINAL PRESENTATION

TEAM 8

Team Members & Roles

LANKA PRASANNA	PROJECT MANAGER
SIDDABOENA JEEVAN SAMMESWAR	SYSTEM ARCHITECT
MEGH SHAH	LANGUAGE GURU
SANDEEP L	SYSTEM INTEGRATOR
KETHAVATH PRANEETH NAYAK	SYSTEM INTEGRATOR
SAI SIVA ROHITH TIRUMALASETTI	SYSTEM ARCHITECT
BATHINI ASHWITHA	TESTER



Introduction

- LiTeC is a high-level imperative general-purpose programming language.
- In the acronym LiTeC, Li stands for LISP, Te stands for TEX, and C stands for C programming language.
- This language is the intersection of procedural and object-oriented programming paradigms.
- Proposed features include TeX Output, Subfiles, and Enumerated comments.

Key features

Language design

- Essential: Variables, data types, expressions, data structures (arrays and matrices), statements, functions, blocks, loops, and standard libraries(proposed) are there in LiTeC.
- Unique: There is a function called TeX, which takes input in the Tex programming language but can also include functions or expressions of LiTeC. This enables us to solve complex problems in LiTeC and call those functions directly in TeX. This feature can be further extended by embedding the LaTeX compiler with LiTeC so that the output of the TeX function results in pdf.

Key features

Language design

- Efficiency: We use prefix expressions inspired by LISP. Since there is no concept of precedence and associativity, these expressions make our language highly efficient.
- OOPs: Our program has subfiles that enable multiple programmers to simultaneously work on the same file. Each subfile has a block of code that can be inherited by subfiles following it. Object-oriented concepts of encapsulation and inheritance can be seen here.

Phases of compiler design

1

Lexical analysis

Converts sequence of characters into tokens.

2

Syntax analysis

Checks syntactical structure by building parse tree.

3

Semantic Analysis

Checks semantical correctness using syntax tree and symbol table.

4

Code Generation

Source code is converted to machine code and is ready for execution.

Key features

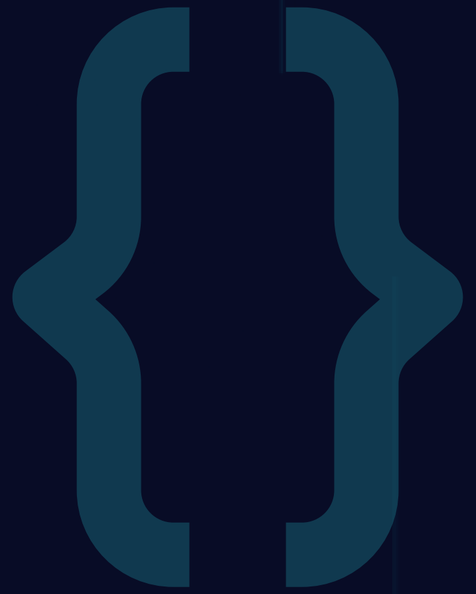
Lexical analysis

- Smallest individual and meaningful elements identified by the compiler are known as tokens.
- High-level source program is converted to a sequence of tokens in this phase.
- We used the lex tool to generate a lexical analyzer.
- Comments and whitespaces are removed.
- Line number is added.
- Our lexer takes LiTeC source code and returns all valid tokens in standard output in this phase.

Key features

Syntax analysis

- In this phase, tokens are taken as input, and output is a parse tree.
- We used the YACC tool as a parser generator.
- We wrote grammar containing a set of rules to recognize strings of our language.
- Our parser outputs syntax errors if there are any unrecognized or missed tokens and prints parsing completed when everything is syntactically correct.



Key features

Semantic analysis

- We implemented a symbol table and Abstract syntax tree.
- We used both the symbol table and AST to do semantic analysis.
- The following semantic checks were included in our code:
  - Re-declaration of variables.
  - Type checking for expressions and function return types.
  - Type checking of function arguments.
  - Undeclared variables.

Examples

An example LiTeC program showing matrix operations.

```
{[
    MainFile
    ([
        declare int globa_variable : 10;
    ])


    int main()
    {
        declare int A[5][5];
        declare int i:0;
        declare int j:0;
        loop(i: 0; i<10 ;i:(+ i 1) )
        {
            loop(j : 0; j < 10; j : (+ j 1))
            {
                A[i][j]: (+ i j);
            }
        }
        declare int B[5][5] : A;

        A : (+ A B);
        B : (* B A);
        A : (* 4 A);
        B : (/ B 7);
        A : (- (+ A B) (/ (* B A) 1));

        return 0;
    }

]}
```

Examples

A part of the program shows how we call the TeX function. Note: fact function is defined in the previous subfile, which is not shown here but is inherited.

```
{[
    SubFile1
    ([
        int mul(int a1 , int a2)
        {
            return (* a1 a2);
        }
    ])

    int main1()
    {
        declare int x : 10;
        declare int y;

        y : (fact(12));
        y : (mul(x, y));

        TeX
        {
            [:(mul(4, (mul(1, 7)))):],
            "Hello",
            [:(fact((mul(x, y)))):],
            "Error"
        }

        return 0;
    }
]}
```


Thank You!