**CUSTOM CALCULATOR LANGUAGE**

A green shield with gold letters and a gold laurel wreath

Description automatically generated

**SEMESTER PROJECT**

**PRESENTED BY**

AHMAD MUSTAFA

TAHIR IQBAL

USAIRA SHAHBAZ

**19 JAN 2025**

**NAMAL UNIVERSITY**

**MIANWALI**

**1. Introduction**

This project is a comprehensive implementation of a simple compiler for a custom programming language integrated into an interactive Integrated Development Environment (IDE). The compiler performs all essential phases, including lexical analysis, syntax analysis, semantic analysis, code generation, and execution. The IDE features a user-friendly GUI built with Python's tkinter, providing functionalities like code editing, error feedback, and visual output display.

The primary objective is to introduce the fundamental concepts of compiler construction and programming language design while offering a platform for learners to experiment and understand compiler workings.

**2. Features**

The project includes the following features:

* **Custom Programming Language**: A basic language with a concise set of keywords, operators, and syntax.
* **Compiler Phases**:
  + Lexical Analysis
  + Syntax Analysis
  + Semantic Analysis
  + Code Generation (Three-Address Code and Assembly-like Code)
  + Execution
* **Symbol Table Generation**: Tracks all identifiers with their attributes (type, scope, etc.).
* **Interactive GUI**:
  + Code editor with syntax highlighting.
  + Output display for errors, results and generated code.
  + Clear, Save, and Run buttons for ease of use.
* **Error Handling**: Real-time feedback for syntax and semantic errors.
* **File Handling**: Save and load code files.

**3. Keywords in the Custom Language**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Keyword** | | **Description** | |
| VAR | | Declares a variable | |  |
| ADD | | Addition operation | |  |
| SUB | | Subtraction operation | |  |
| MUL | | Multiplication operation | |  |
| DIV | | Division operation | |  |
| PRINT | | Outputs a value to console | |  |
| COS | | For cos theta | |  |

SIN For sin theta

TAN For tan theta

POW Power operation

LOG Logarithmic operation

**4. Custom Language Syntax**

**Variable Declaration:**

VAR x = 10;

**Arithmetic Operations:**

ADD 5 3

MUL x 2

**Print Statement:**

PRINT x;

**5. Phases of the Compiler**

**5.1 Lexical Analysis**

* **Function**: Scans the source code and breaks it into tokens.
* **Output**: A list of tokens with their types (e.g., identifier, keyword, operator).
* **Example**:
  + Input: VAR x = 10;
  + Tokens: [('VAR', 'keyword'), ('x', 'identifier'), ('=', 'operator'), ('10', 'number')]

**5.2 Syntax Analysis**

* **Function**: Constructs a parse tree to validate the structure of the code.
* **Output**: Abstract Syntax Tree (AST).
* **Example**:
  + Input: x = ADD(5, 3);
  + AST: Assignment -> [Identifier, FunctionCall -> [Arguments]]

**5.3 Semantic Analysis**

* **Function**: Checks for semantic errors (e.g., type mismatches, undeclared variables).

**5.4 Code Generation**

* **Function**: Converts the AST into three-address code (TAC) and assembly-like code.
* **Output**:
  + TAC: t1 = 5 + 3; x = t1;
  + Assembly: LOAD 5; ADD 3; STORE x;

**5.5 Execution**

* **Function**: Executes the generated code and displays the output.
* **Output**: The result of the program.

**6. GUI Design**

**6.1 Code Editor**

* A text area for writing the source code.
* Features are syntax highlighting for keywords and error indicators.

**6.2 Buttons**

|  |  |
| --- | --- |
| **Button** | **Functionality** |
| Run | Compiles and executes the code. |
| Clear | Clears the code editor and output. |
| Save | Saves the current code to a file. |
| Load | Loads code from a file into the editor. |

**6.3 Output Sections**

* **Lexical Analysis Output**: Displays the tokens.
* **Syntax Analysis Output**: Shows the parse tree or AST.
* **Semantic Analysis Output**: Provides success or error messages.
* **Code Generation Output**: Shows TAC and assembly code.
* **Execution Output**: Displays the program’s result.

**7. Complete Functionality of Buttons**

* **Run**: Executes all compiler phases sequentially and displays results in the respective output sections.
* **Clear**: Clears all input and output sections.
* **Save**: Saves the code editor content to a .txt file.
* **Load**: Loads a code file into the editor.

**8. Example Program and Output**

**Input Code:**

VAR x = 5

VAR y = 10

VAR z = ADD x y

PRINT x

**Output:**

1. **Lexical Analysis**:

[('VAR', 'keyword'), ('x', 'identifier'), ('=', 'operator'), ('5', 'number'), ...]

1. **Syntax Analysis**:

Assignment -> [Identifier, FunctionCall -> [Arguments]]

1. **Semantic Analysis**:
2. **Code Generation**:

**TAC:**

* + t1 = x + y;

x = t1;

**Assembly:**

* + LOAD x;
  + ADD y;

1. **Execution Output**:

Output:15

**9. Conclusion**

This project serves as a hands-on introduction to compiler design and programming language development. By integrating a fully functional GUI, it provides an interactive and educational platform for users to learn the intricacies of compilers and programming language syntax.