Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Computer Programming

Individual Work

Author:
Mihaela Untu
std. gr. FAF-232
Andi Blindu
std. gr. FAF-233

Verified:
Alexandru Furdui

Chișinău 2023

Theory Background

A Domain-Specific Language (DSL) is a computer language that's targeted to a particular kind of problem, rather than a general purpose language that's aimed at any kind of software problem. Domain-specific languages have been talked about, and used for almost as long as computing has been done.

The Task

Hard - create your own simple calculator Domain-Specific Language (DSL)

Task Constraints:

- YOU MUST USE C OR C++ for the implementation.
- You must implement a lexer, a parser, and an interpreter for your DSL.
- Your interpreter should parse the tree generated by the parser, not just the tokens produced by the lexer.
- *Variable Declaration*: Each variable declaration must specify its data type explicitly(e.g., 'var int x', 'float y = 9.8').
- *Arithmetic Operations*: Your DSL must support basic arithmetic operations, such as addition, subtraction, multiplication, and division.
- *Control Structures*: Implement at least one control structure, such as conditionals (e.g., if statements) and one loop (e.g., for or while loops) in your DSL.
- *Error Handling*: Your DSL should provide clear error messages for syntax and runtime errors, making it user-friendly.
- *Interactive Mode*: Implement an interactive mode where users can enter DSL code line by line(you can use external text file for code input).
- *Modular Design*: Organize your DSL implementation into separate modules or components for the lexer, parser, and interpreter to maintain a clean and maintainable codebase.
- *Source Code*: Your codebase should be organized into multiple files to improve clarity and ease of understanding. Consider dividing your code into separate modules or components. The entire codebase should be uploaded to a GitHub repository as you won't be able to upload multiple files on ELSE.

Technical implementation

The code in C++: https://github.com/Individual-Work-DSL.git

Keycomponents of this DSL

1. The **Lexer** class tokenizes the input string into a sequence of tokens, where each token has a type (TokenType) and a value. Tokens represent different elements of the programming language, such as integers, decimals, operators, keywords, variables, and control flow constructs.

2. The **enum TokenType** defines different types of tokens that the lexer can recognize. Each type corresponds to a specific element in the DSL, such as integers, decimals, operators, control flow keywords, etc.

3. The **Token struct** represents an individual token with a type and a value. The Lexer produces a stream of these tokens from the input string.

4. The **Parser class** processes the sequence of tokens generated by the Lexer and generates a logic tree based on the structure and semantics of the DSL.

5. The **Interpreter class** interprets the tree of tokens generated by the Parser and performs actions based on the structure and semantics of the DSL. It includes methods for handling variable declarations, mathematical operations, control flow statements (if, while, loop), and printing values.

6. The DSL supports various operations, including variable declaration (INTDECLARE and DECDECLARE), mathematical operations (PLUS, MINUS, MULTIPLY, DIVIDE), control flow constructs (IF, WHILE, LOOP), printing (PRINT), and stopping the program (STOP).

7. Examples of valid input expressions in this DSL include:

```
number a : 3 repeat 3 start print a end multiply a
   to a while [a < 10] start add 1 to a print a end
   number b:10 stop
```

8. The DSL includes some basic **error handling**, such as checking for variable redeclaration and handling invalid expressions.

9. The **readFromFile function** allows users to load DSL code from a file. The main function includes a simple menu with options to load code from a file and compile it.

10. The DSL is designed for simple interactive use, where users can input DSL code directly or load it from a file. The menu-driven interface provides options for interacting with the DSL.

Syntax Keyword Meanings

Table 1: Meanings of Syntax Keywords

Syntax Keyword	**Meaning**
INTEGER	Integer input, e.g., 123
DECIMAL	Decimal input, e.g., 12.34
PLUS	Addition operator, represented by the keyword **add**.
INTDECLARE	Integer variable declaration, represented by the keyword **number**.
DECDECLARE	Decimal variable declaration, represented by the keyword **decimal**.
MINUS	Subtraction operator, represented by the keyword **subtract**.
MULTIPLY	Multiplication operator, represented by the keyword **multiply**.
DIVIDE	Division operator, represented by the keyword **div**.
LOOP	Loop construct, represented by the keyword **repeat**.
WHILE	While loop construct, represented by the keyword **while**.
IF	If statement, represented by the keyword **if**.
START	Start block, represented by the keyword **start**
END	End block, represented by the keyword **end**.
VARIABLE	Variable name, defined by the regular expression `[_a-zA-Z][_a-zA-Z0-9]*`.
TO	Keyword **to**, used in looping constructs.
FROM	Keyword **from**, used in looping constructs.
EQUALITY	Relational operators (**==**, **!=**, >, <, **>=**, <=).
LOGIC	Logical expressions enclosed in **square brackets**, e.g., [**a** > **b**].
ASSIGN_INT	Integer variable assignment, represented by **-?\d+** pattern
ASSIGN_DEC	Decimal variable assignment, represented by **: -?\d+\.\d+** pattern.
SPACE	Whitespace.
STOP	Stop token, indicating the end of the program.
PRINT	Print statement, represented by the keyword **print**.
INVALID	Invalid or unrecognized token.

Instructions on How to Write a Sample Program

1. **Define your program**.

For example, create a program that declares two variables, adds them, and then prints the result.

```
number a : 5
number b : 10
start
add a to b
print a
end
stop
```

2. **Save the program in a text file**

Save the program in a text file with a .txt extension, for example, sample-program.txt.

3. **Compile and run**

Compile the C++ program, run it, and choose option 1 to load the program from the file and option 2 to compile and execute it. The DSL parser will process the program and provide the output.

Program Examples

Example 1: **Error-Free Program**

```
number a : 5 repeat 3 start
    print a
end
stop
```

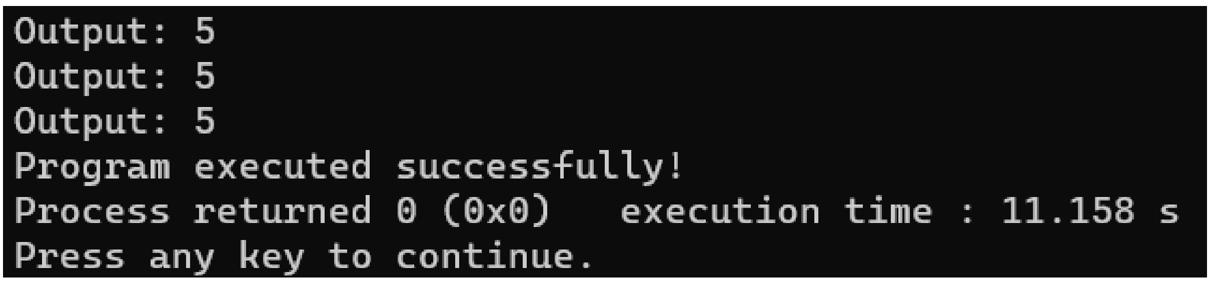

Figure 1: Error-Free Program

Example 2: **Program with Errors**

```
number b : 10 repeat 2 start
    print b
end
multiply c to b  // Error: 'c' is not declared.
stop
```

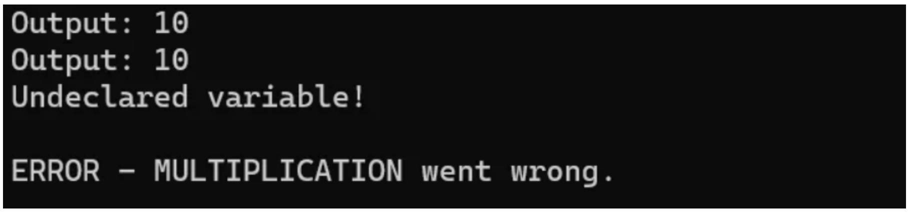

Figure 2: Program with Errors

Example 3: **Error-Free Program with If Statement**

```
number x : 8
number y : 10
if [x < y] start
    print y
end
stop
```

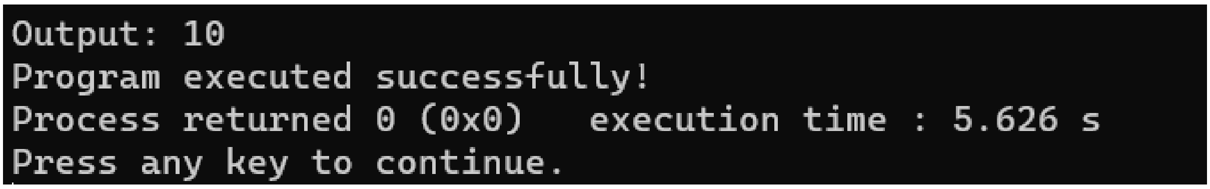

Figure 3: Error-Free Program with If Statement

Example 4: **Program with Error in Logic Expression**

```
number p : 15
number q : 10
if [p = q] start
    print p  // Error: Logic expression is not true.
end
stop
```

```
ERROR - IF went wrong.

Process returned 0 (0x0)   execution time : 4.434 s
Press any key to continue.
```

Figure 4: Program with Error in Logic Expression

Example 5: **Error-Free Program with Loop**

```
number n : 3
repeat 3 start
    print n
end
stop
```

```
Output: 3
Output: 3
Output: 3
Program executed successfully!
Process returned 0 (0x0)   execution time : 4.620 s
Press any key to continue.
```

Figure 5: Error-Free Program with Loop

Example 6: **Program with Error in Loop Statement**

```
loop start
    print "Error"  // Error: 'loop' statement requires a
       numeric value.
end
stop
```

```
Syntax ERROR!

Process returned 0 (0x0)   execution time : 5.089 s
Press any key to continue.
```

Figure 6: Program with Error in Loop Statement

Bibliography

1. https://youtube.com
2. https://martinfowler.com
3. https://study.com