Al Imam Mohammad ibn Saud Islamic University
College of Computer and Information System
Computer Science Department

Compilers Project

Children pseudocode editor

Prepared by:

Atheer Alamri
Reem AlLuhaidan
Reem AlRayes
Salma Fahmy

Supervised By:

Dr. Amal AlSaif

December 2, 2021

Table of Contents

List of Figures

1. Lexical Analyser

Lexical Analysis is the first phase of the compiler also known as a scanner. It converts the High-level input program into a sequence of Tokens [1]. can be implemented with the Deterministic finite Automata (DFA) or Non-Deterministic finite Automata (NFA) .This phase provides the Lexical analyser for our language.

1.1. Regular Expression

Regular expressions (RE) are patterns used to match character combinations in strings [2]. This section shows the RE for our language.

```
primitiveTypes = double|int|float|char|boolean
keyWordsIf = if|else|else if
keyWordsLoop = for|while|do
keyWords = if|else|else if|for|while|do
identifiers = [a-zA-Z][a-zA-Z0-9_]*
string = ^(\".*\")$
charecter = ^(\'.\')$
assignmentOperator = [=]
arithmeticOperator = \\+|\\-|\\*|%
unaryOperator = ([\\+]{2}|[\\-]{2})
relationalOperator = <=|>=|==|=|>|<|!=
conditionalOperator = [\\&\\|]{2}
symbol = [\\(\\)\\{\\}\\[\\]\\.\\,\\:\\'\\\"]
semicolon = [;]
integerNumbers = ([0-9]+|\\-[0-9]+)
floatNumbers = ([0-9]+\\.[0-9]+|\\-[0-9]+\\.[0-9]+)
boolean2 = true|false
startBlock = "[:]"
endBlock = EndBlock
program = program
end = End
```

2. Syntax Analyser

It checks the syntactical structure of the given input by building a Parse tree or Syntax tree. The parse tree is constructed by using the pre-defined Grammar of the language and the input string. If the given input string can be produced with the help of the syntax tree (in the derivation process), the input string is found to be in the correct syntax. if not, the error is reported by the syntax analyser [3]. This phase provides the Lexical analyser for our language.

2.1. Context Free Grammar

A Context Free Grammar (CFG) is a set of recursive rules used to generate patterns of strings [4]. This section shows CFG for our language.

Pseudocode: Children Editor

Program: Develop a children pseudocode editor
Output: Sample table of the tokens for the code and described the lexical errors via meaningful messages with line number.

1. ***Program*** → program Inside **Program** | Inside **Program** | End
2. ***Inside*** → Initialization | Declaration | Selection | Expression | Loop
3. ***Initialization*** → PrimitiveTypes identifiers AssignmentOperator Value Semicolon
4. ***Value*** → IntegerNumbers | FloatNumbers | Boolean | Charecter | String
5. ***Declaration*** → PrimitiveTypes identifiers Semicolon
6. ***Selection*** → KeywordsIf Selection2
7. ***Selection2***→ Symbol S Symbol StartBlock Inside EndBlock Selection3
8. ***Selection3***→ Selection | e | StartBlock Inside EndBlock
9. ***S*** → Expression RelationalOperator Expression S2
10. ***S2*** → ConditionalOperator S | e
11. ***Expression*** → Value | identifiers Expression2
12. ***Expression2*** → ArithmeticOperator identifiers | RelationalOperator identifiers | UnaryOperator | e
13. ***Expression2*** → ArithmeticOperator Value | RelationalOperator Value | UnaryOperator | e
14. ***Loop*** → KeywordsLoop Symbol Loop2
15. ***Loop2*** → Initialization Expression Semicolon Expression Symbol StartBlock Inside EndBlock | S Symbol StartBlock Inside EndBlock

3. Code

3.1. Lexical and Syntax analysis code

The classes used [5], are as follows:

- Demo “Start interface”
- Analyser “Lexical and Syntax analysis class”
- Token
- LR1Parser
- Parser
- AugmentedFirstAndFollow
- PrettyPrinter

3.2. Regular expression to Deterministic finite automata code

Convert RE to DFA by creating and using a syntax tree [6]. The classes used, are as follows:

- ConvertRegextoDFA “class”
- SyntaxTree.
- BinaryTree.
- Node.
- LeafNode.
- DfaTraversal.
- State.

4. Screenshots

4.1. The output of Lexical and Syntax analyser.

4.1.1. CFG file

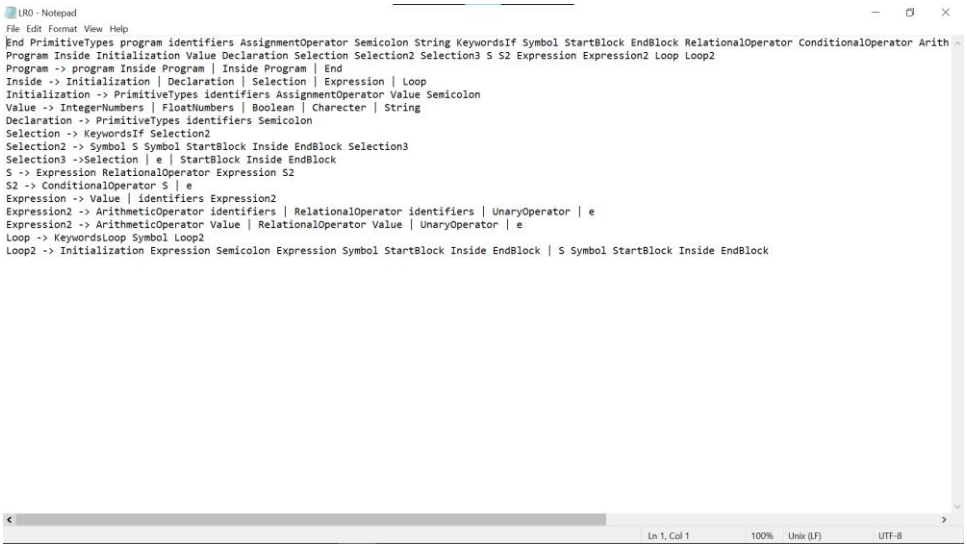


Figure 1: A text file of the CFG

4.1.2. Start interface

Figure 2: The Start GUI of the children pseudocode editor

4.1.3. The output of the correct input code

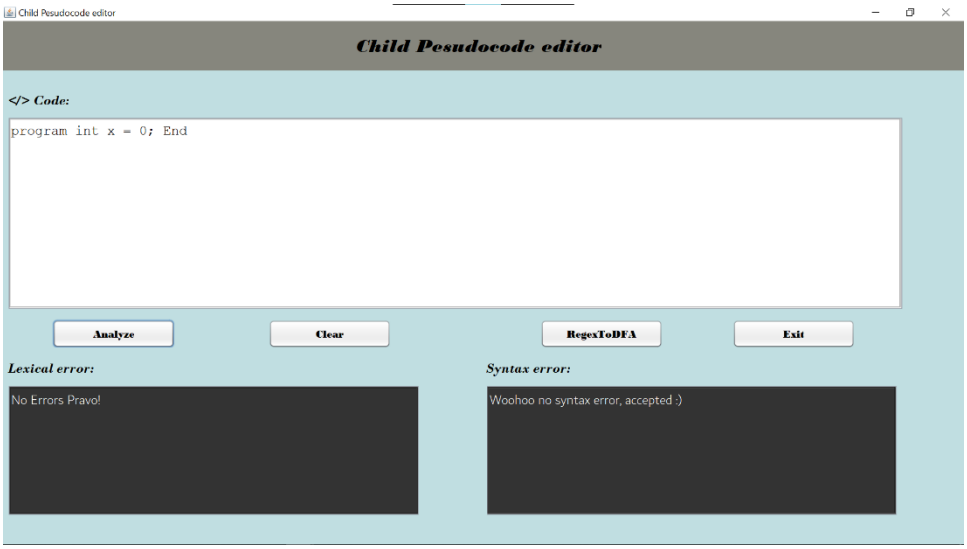


Figure 3: The output interface of the lexical and syntax analysis phase for a correct input code

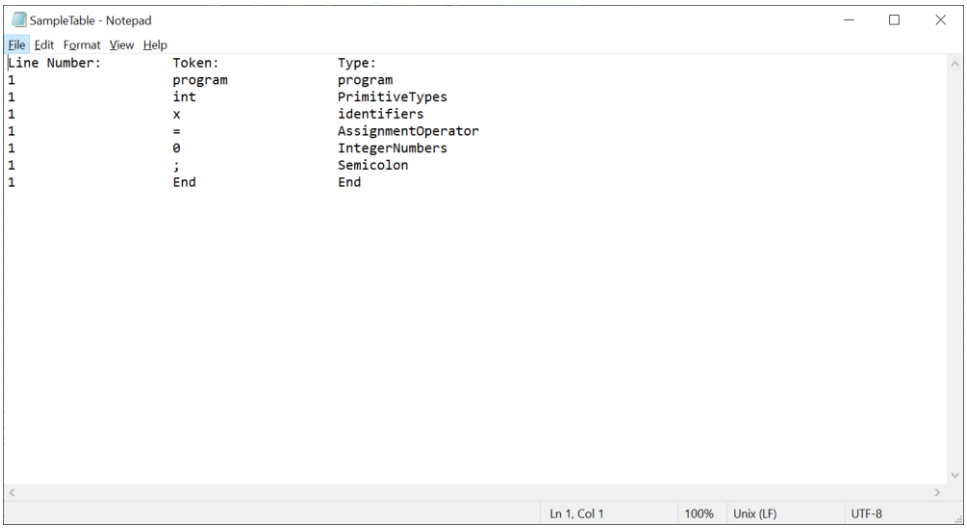


Figure 4: The text file of the symbol table of a correct token

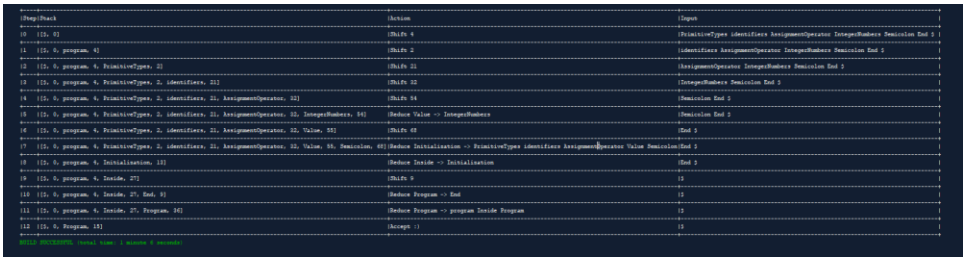


Figure 5: The output actions of a LR(1) Parser of a correct input

4.1.4. The output of the incorrect input code

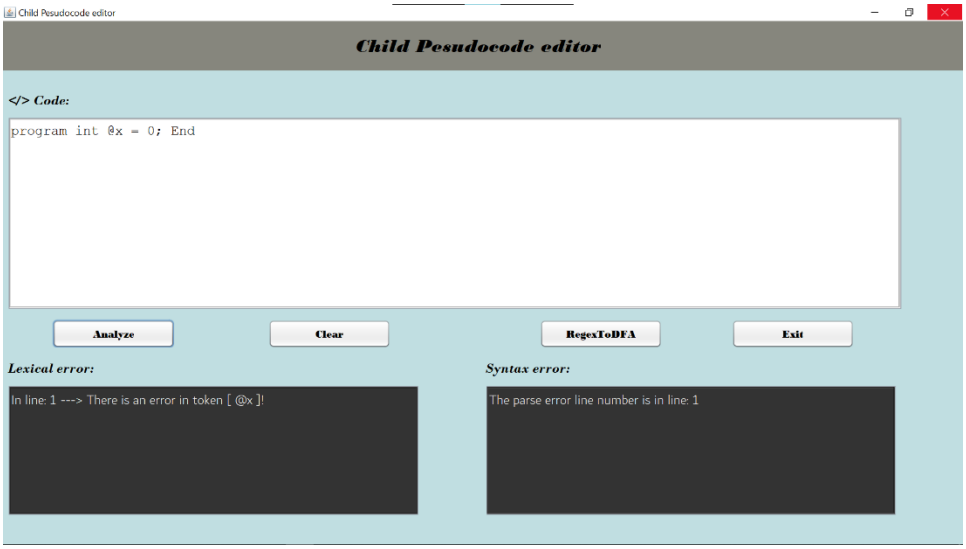


Figure 6: The output interface of the lexical and syntax analysis phase for an incorrect input code

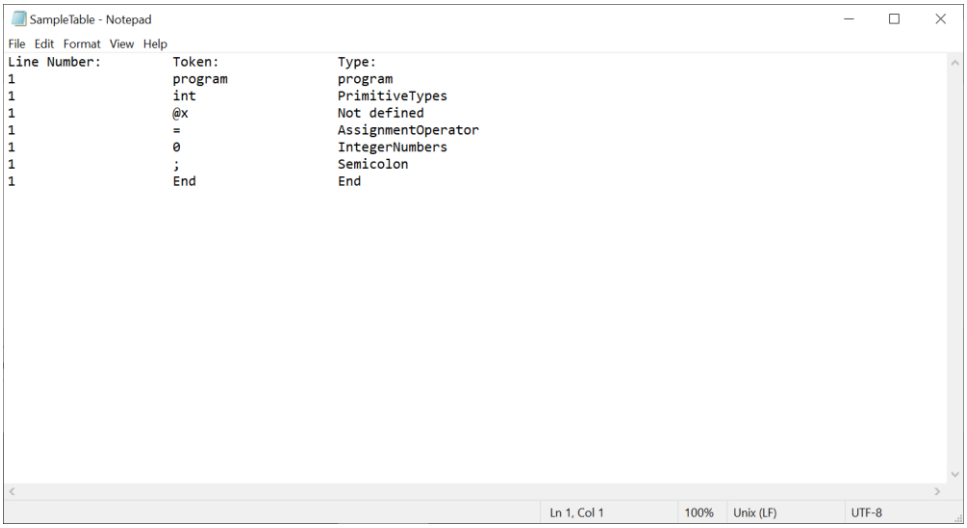


Figure 7: The text file of the symbol table of an incorrect token

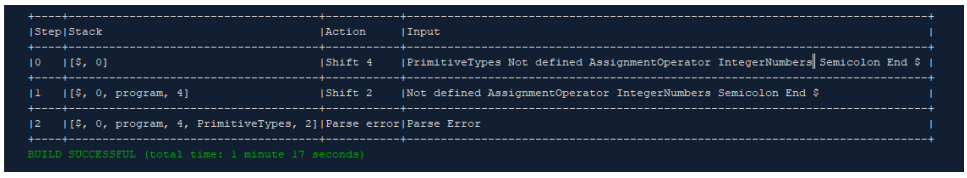
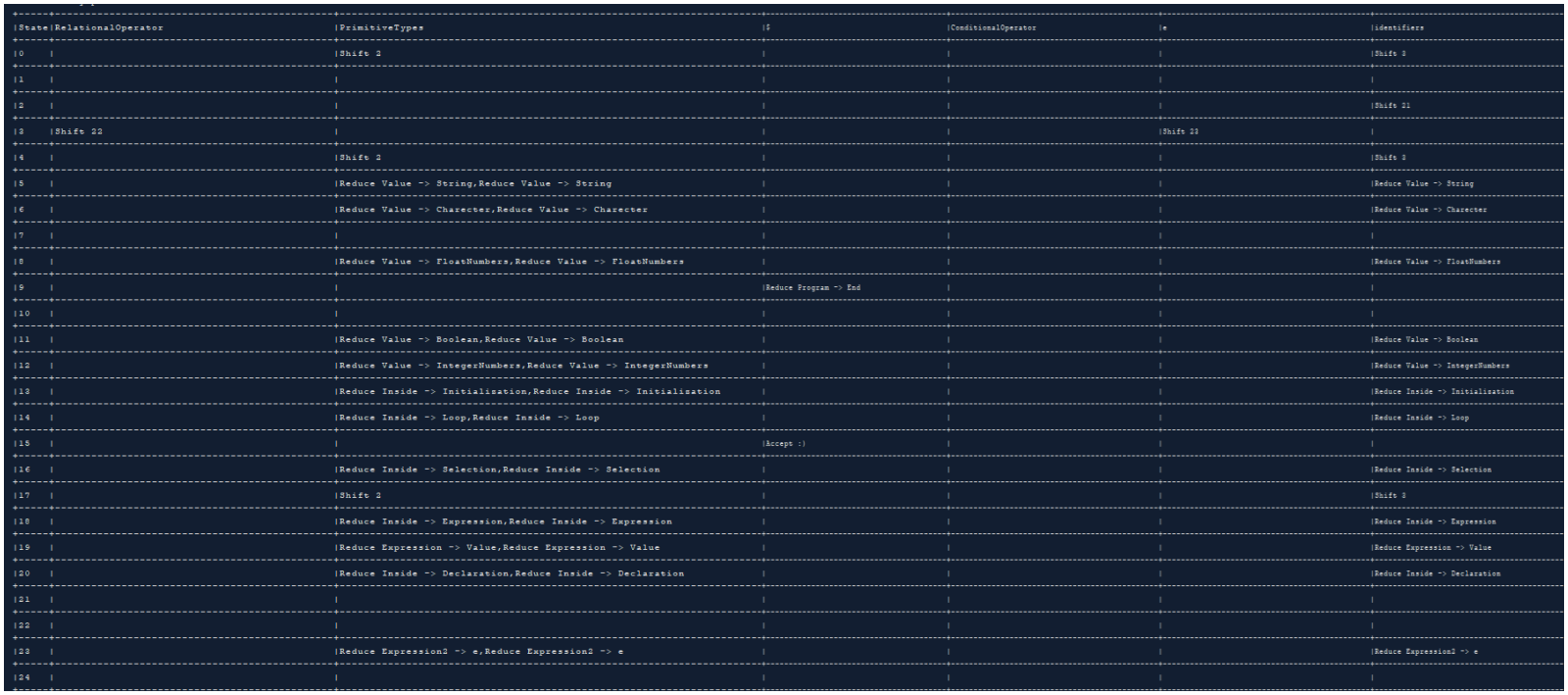


Figure 8: The output actions of a LR(1) Parser of an incorrect input

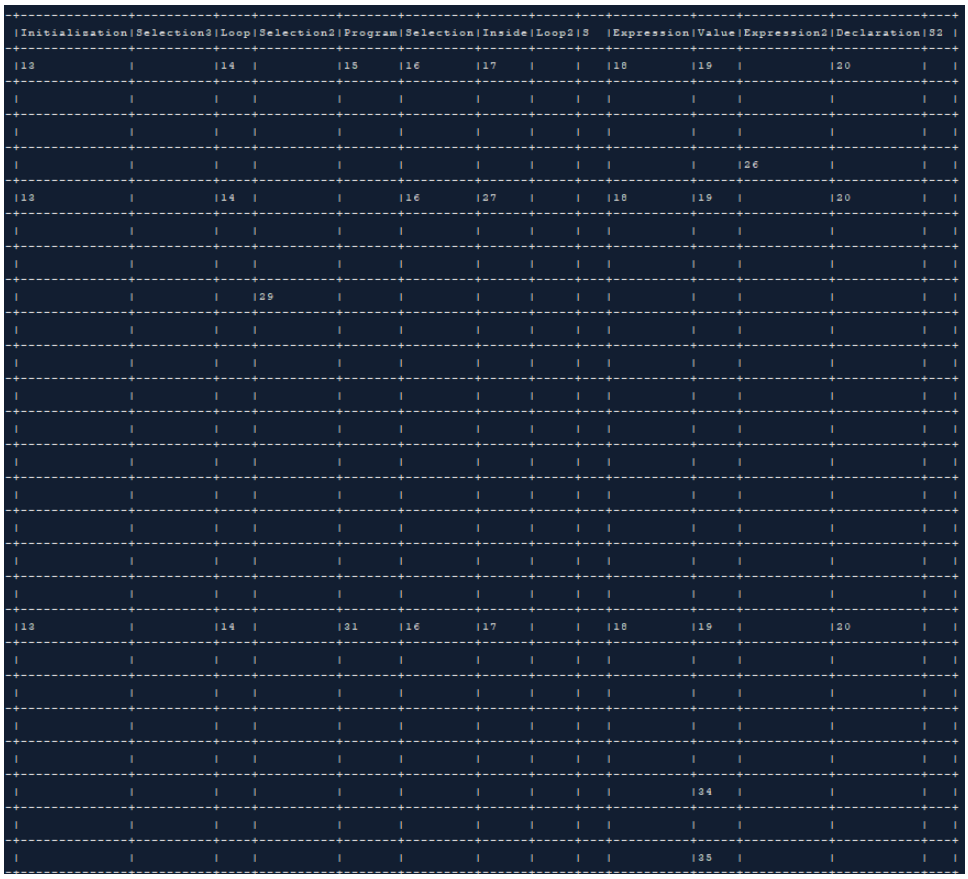
4.1.5. The parsing tables

The parser table contains a huge state that reached 186 states and 22 terminal symbols that can't be included all in a document.



State	RelationalOperator	PrimitiveTypes	$	ConditionalOperator	e	identifiers
0		Shift 2				Shift 3
1						
2						Shift 21
3	Shift 22				Shift 23	
4		Shift 2				Shift 3
5		Reduce Value -> String,Reduce Value -> String				Reduce Value -> String
6		Reduce Value -> Character,Reduce Value -> Character				Reduce Value -> Character
7						
8		Reduce Value -> FloatNumbers,Reduce Value -> FloatNumbers				Reduce Value -> FloatNumbers
9			Reduce Program -> End			
10						
11		Reduce Value -> Boolean,Reduce Value -> Boolean				Reduce Value -> Boolean
12		Reduce Value -> IntegerNumbers,Reduce Value -> IntegerNumbers				Reduce Value -> IntegerNumbers
13		Reduce Inside -> Initialization,Reduce Inside -> Initialization				Reduce Inside -> Initialization
14		Reduce Inside -> Loop,Reduce Inside -> Loop				Reduce Inside -> Loop
15			Accept :)			
16		Reduce Inside -> Selection,Reduce Inside -> Selection				Reduce Inside -> Selection
17		Shift 2				Shift 3
18		Reduce Inside -> Expression,Reduce Inside -> Expression				Reduce Inside -> Expression
19		Reduce Expression -> Value,Reduce Expression -> Value				Reduce Expression -> Value
20		Reduce Inside -> Declaration,Reduce Inside -> Declaration				Reduce Inside -> Declaration
21						
22						
23		Reduce Expression2 -> e,Reduce Expression2 -> e				Reduce Expression1 -> e
24						

Figure 9: Part of the actions table



Initialisation	Selection3	Loop	Selection2	Program	Selection	Inside	Loop2	S	Expression	Value	Expression2	Declaration	S2
13		14		15	16	17			18	19		20	
											26		
13		14			16	27			18	19		20	
			29										
13		14		31	16	17			18	19		20	
										34			
										35			

Figure 10: Goto Table

4.2.The output of RE to DFA function

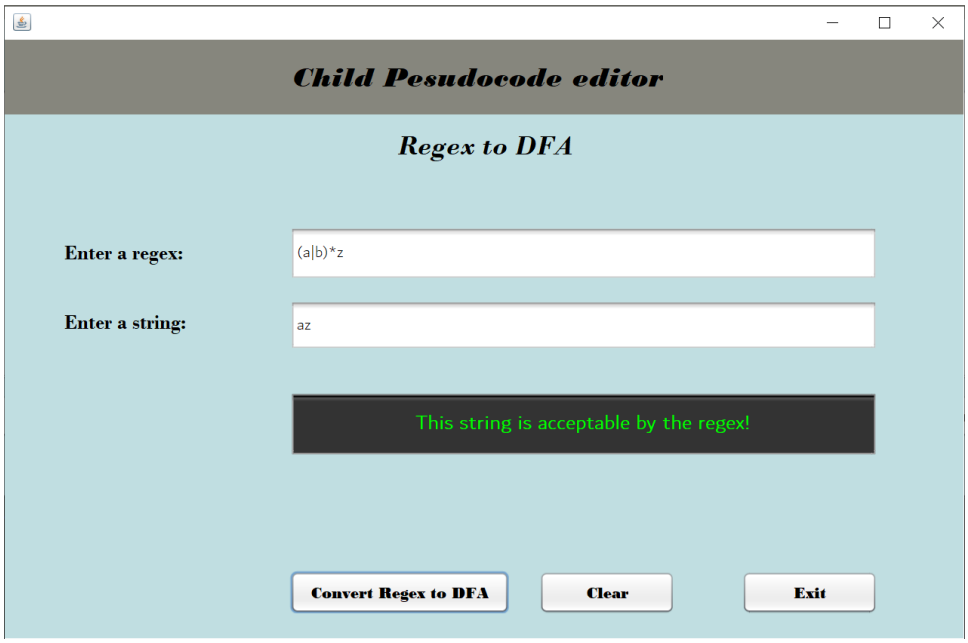


Figure 11: The output for the string is acceptable

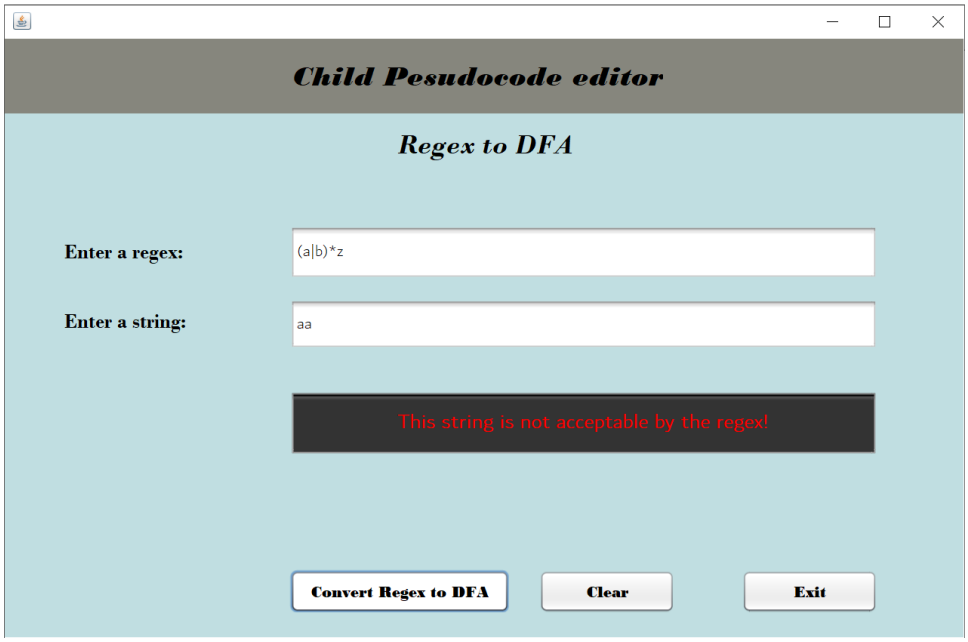


Figure 12: The output for the string is not acceptable

References

[1] "Introduction of Lexical Analysis - GeeksforGeeks," 13 July 2015. [Online]. Available: https://www.geeksforgeeks.org/introduction-of-lexical-analysis/. [Accessed 5 November 2021].

[2] "Regular expressions - JavaScript | MDN," 22 October 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions. [Accessed 5 November 2021].

[3] "Introduction to Syntax Analysis in Compiler Design - GeeksforGeeks," 22 September 2015. [Online]. Available: https://www.geeksforgeeks.org/introduction-to-syntax-analysis-in-compiler-design/. [Accessed 1 December 2021].

[4] "Context Free Grammars | Brilliant Math & Science Wiki," 5 November 2021. [Online]. Available: https://brilliant.org/wiki/context-free-grammars/. [Accessed 5 November 2021].

[5] "CompilerDesignLab/Parser_Library at master · PalAditya/CompilerDesignLab," 2 December 2021. [Online]. Available: https://github.com/PalAditya/CompilerDesignLab/tree/master/Parser_Library. [Accessed 2 December 2021].

[6] alirezakay, "RegexToDFA," w Nov 2018. [Online]. Available: https://github.com/alirezakay/RegexToDFA.