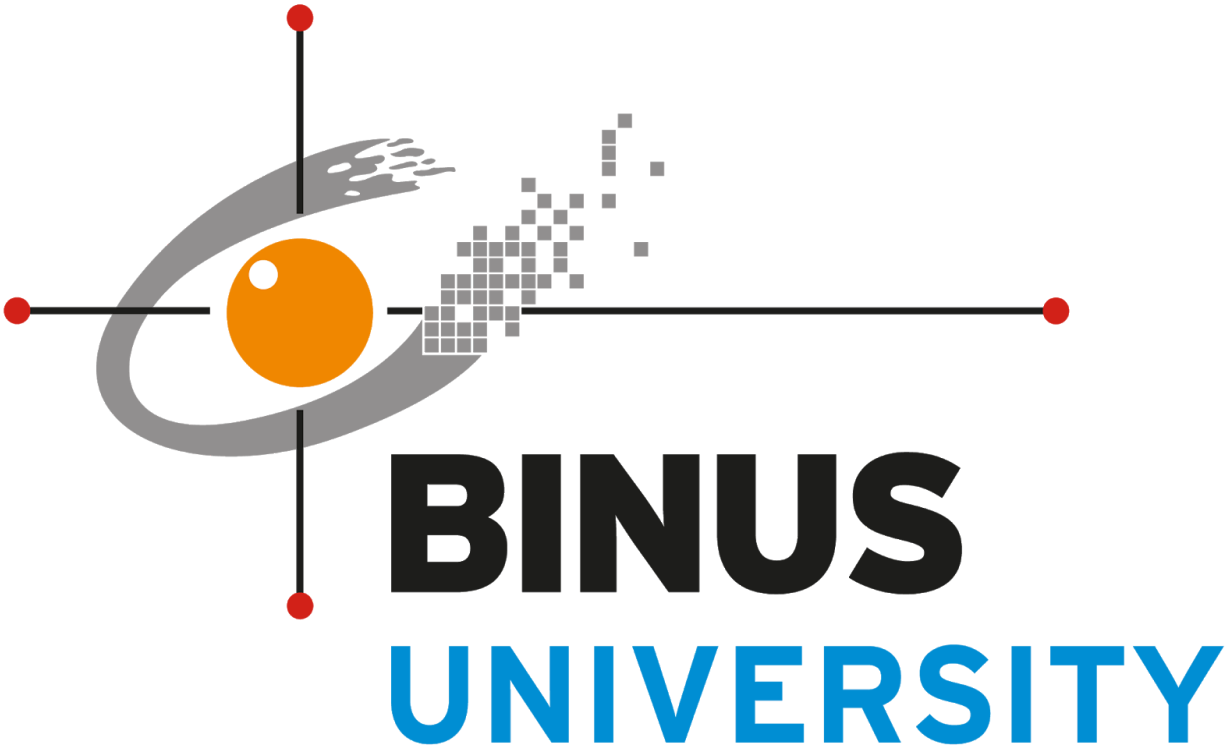
Final Project Report

E.R.I.C Programming Language



Rowin Faadhilah Roestam Moenaf (2301944084)
Michael Joseph (2440116965)
Calvin Scorpiano Halim (2301915374)
Mileno Valdo Elvano (2301929505)

Introduction

Background

This project is done by a team of 4 students for Analysis of Algorithms class. The class focuses on different algorithms that can be used for solving common programming problems, it also covers the principles of compiler design. What will be shown in this report is the knowledge that was gained from the lessons in the form of a final project. The final project in this case would be a new programming language that the group has designed.

Project Specification

The goal of this project is to build algorithms that would solve particular programming problems or to apply the principles of compiler design in practical problems. The project is designed to help develop the team's coding knowledge and skills. For this project the group had chosen to tackle compiler design problems. The problem that was chosen is building our own programming language.

Related Works

The inspiration of E.R.I.C programming language came from another language known as HolyC which was written by a schizophrenic computer programming, Terry A. Davis.

Link: https://harrisontotty.github.io/p/a-lang-design-analysis-of-holyc
Link: https://github.com/jamesalbert/HolyC-for-Linux

Implementation

Problem

In this project the task was to create a "cool" programming language by using the knowledge gained from classes. Programming languages are tools that are used to write commands so that a computer would be able to understand it. The motivation that pushed us to create this project is that programming languages are the main building blocks of code, although lessons provide us with information on programming language, creating one would be a challenge that might increase the group's experience. Attempting and understanding how a programming language was made might help the group to become better programmers.

Based on the group's experiences, this problem looks like a very challenging problem to solve. To build a new programming language, a lexer algorithm is needed, as well as a parser. There are many methods that can be used to build the lexer and the parser. It was decided that the group will limit the use of modules when building the lexer and parser. Instead lists were used to compute the lexer algorithm.

Design

The programming language will be built using Python. Python has modules that support building your own programming language but the group wanted a challenge and decided that those modules will not be used. Built-in functions in python are also helpful to create the programming language hence why Python was used. Since the specifications of this project require the final product to be in the form of an application, a module known as Tkinter was used to create a GUI.

The lexer algorithm is designed to take in tokens from a file. The file contains the commands for the programming language. Each character in the file will be split and checked through a loop at the end of a loop they will be appended into a string which will form the tokens. If conditions are set to locate specified tokens. If a token is found it will be appended into a list of tokens, and that is where all tokens from the command will be collected.

After the lexer had gone through all the characters in the file. It will return the list of tokens that was created during the process. The list is then used to be parsed. The parser function also contains different if conditions that would figure out what needs to be done depending on the tokens in the list. By using string slicing and list indexing the parser will be able to pinpoint each token. After finding out what token the parser is handling, they will run the codes that are set to that specific token.

The GUI that was made to support this program only has a text box that allows the user to write their codes and a button that will allow them to run the code. The output will still be displayed in the command line of the original IDE since there were issues trying to output the result in the GUI. The button that allows the user to run also makes sure that the code that the user has inputted will be saved in the file that will be processed by the lexer.

Complexity

Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
openFile	13	17	1	5	1
lex	20	164	⚠ 73	⚠ 144	1
ericPrint	167	175	4	9	1
evaluate	177	178	1	2	1
assign	180	181	1	2	2
getVariable	183	189	2	6	1
getInput	192	194	1	3	2
getDo	196	197	1	2	1
parse	199	374	⚠ 82	⚠ 150	1
savefile	478	480	1	3	0
run	484	491	1	8	0

Judging by the image below it shows that the lexer function has a very high complexity as well as the parser function. This is due to the amount of nested if statements in a for loop that are needed to sort the tokens from a certain file properly. As for the parser this is also because of the amount of conditions that are needed so that the parser is able to determine the right process for each token.

Evaluation

Implementation Details

At the start of the program the modules that are required should be imported and in this program's case, importing Tkinter is crucial for the GUI. Global variables are initialized after, these global variables are what would contain the tokens, as well as data regarding eric variables (creating variables with eric). Since this program uses files functions to open and save the file are created as well.

The lexer function has one argument which is the filename that it is going to read. It has a few variables that need to be declared and these variables decide how the lexer is going to store the tokens. After the variable declaration a for loop is used to iterate through the file by a character each. Each character will be added into a string which will form the token. If statements follow and they declare which token to add into one of the variables that is declared in the function which is the list of tokens. Once the lexer function is done iterating through the file, the list of tokens would be returned by the function.

The parser function has one argument which is tokens. It takes the list of tokens that was returned by the lexer function and begins iterating through it using a for loop. The parser function is equipped with if statements that checks what set of tokens are inside the list. The order of the token matters as well as it will determine what the parser would do. For example if the parser was to receive instructions to print out a string the order of tokens in the list should satisfy the if statements in the parser function. An example would be:

List of tokens: [“PRINT”, “STRING:WOMBAT”]

Now with the example above, the parser would recognize the token print and it will check the next index of the list and it will find a string. After finding this it will format the “STRING:WOMBAT” and slice it so that only WOMBAT will be printed out by the code.

Functions such as ericPrint and getDo or getVariable are functions that will help the parser function. These functions are what determines the next steps of what the parser function would compute.

The E.R.I.C Programming Language so far is able to compute common functions that are often used in programming, these include:

- Print
- Input
- Declaring Variables
- Arithmetic Operations
- For loops
- If statements

Data Sets

```
prt "wombat"

prt 10

prt 1+1

$c = 1
prt $c

$s = "value"
prt $s

$a = 10+10
```

```
prt $a

$name = in "Enter your name: "
prt $name

if 10==10 do
        prt "Venti"
fin

if 10>10 do
        prt "Venti"
fin

for 5 do
        prt "pls"
fin

for $c do
        prt "hello"
fin

for 1+5 do
        prt "six"
fin
```

These are the data that was used to test the program. Each line tests a different aspect in the program. What can be seen here is printing various types of data eg. String, Integers, Expressions as well as Variables. Declaring variables can also be seen in this data set. If statements as well for loops are being showcased.

Analysis and Results

After several testing there were errors that came up in the program but eventually it was fixed. Using the data set above outputs what we expected. By using prt the program will print out the element to the next of the print, so the statement ‘prt "wombat"’ will print out “wombat” in the command line. In E.R.I.C variables are declared by adding a “$” before declaring the name of the variable. To keep the value of the variable, a dictionary is used. The value of the variable will always be a string, hence to return an int, data type converting should be done beforehand. The input ‘$c = 1’ will create a variable with the name “$c” that has a value of 1, this can be checked by using prt $c which will return the value of 1. The if statements works just like python except it is limited by using the keywords “do” and “fin”. The keyword “do” lets the program

know that if the condition passes, then the code between “do” and “fin” should be run. Running the input in the data set will print out the string “Skyward Sonnet” because the condition of the if statement passes. The for loop will loop for the number of times that it is declared to. It can read numbers, variables as well as expressions. An example using the dataset above “for 10 do” the for loop will loop for 10 times running the code between “do” and “fin”.

There was an attempt to create a useful programming code to demonstrate the capabilities of E.R.I.C such as creating a factorial code but E.R.I.C still lacks the functionalities for such codes. It is still unable to update variable values by using the same variable and adding it by an expression.

Discussion

At the beginning of the project it was tough to decide on what to do. There were many ideas that were considered but in the end it was decided the creating a programming language would be the group’s final project. After searching online and finding out about similar projects such as the HolyC, creating a programming language sounds like an interesting challenge that the group wanted to tackle. Creating E.R.I.C was a fun and resourceful experience as there are many new things that can be learned during the entire journey of the project. It was taken upon ourselves to make it a bigger challenge by not using lexer modules in python. It was a risk that the group decided to take, and in the very end it turned out to be a great learning experience for everyone in the team.

Conclusion and Recommendation

In conclusion, the programming language that we made satisfied what was expected. E.R.I.C was able to properly run common functions that most programmers use in other programming languages without fail. The lexer function that was created was able to pinpoint every single token from the file that was used. The parser function was also able to perform how it was intended. It reads the tokens and carefully selects the right commands to perform by making use of the string slices as well as the list indexing. Although the complexity of this program is relatively quite high due to the abundant amount of nested if statements, it is able to perform the E.R.I.C codes quite fast.

A recommendation for further developing E.R.I.C is to attempt on reducing the complexity by grouping the nested if statements into a function. There are also more functions that can be added into the programming language such as making comments, or while loops. So far the syntax that we are using is not as simple as other programming languages, we aim to make it simple so programmers can focus more on understanding how E.R.I.C works rather than mastering how to use E.R.I.C. Another thing that can be improved is to make the GUI look more

presentable as well as putting the output in the GUI instead of using the command line of another compiler.

Manual

prt

prt is used to print the various data types that are available in E.R.I.C

```
prt 10
prt 10+2

Output:
10
12
```

```
prt "some strings"

Output:
some strings
```

$

The dollar sign is used to specify the start of declaring a variable name. The variable function is still limited to only declaring variable, it is still unable to update its value using the variable itself.

```
$name = "Rowin"
```

in

in is used to declare an input variable.

```
$name = in "Enter your name: "
```

Arithmetic Operations

Arithmetic operations are declared when an expression is identified. An expression will be identified when one of the symbols eg. +, -, /, * are found in a token.

```
prt 2+2
prt 1+1

Output:
4
2
```

For loops

For loops are declared with the syntax “for *number* do” and the code for the loop will be limited by “fin” so the code that will be runned by the loop will only be between do and fin. The number serves as how much loops will be repeated.

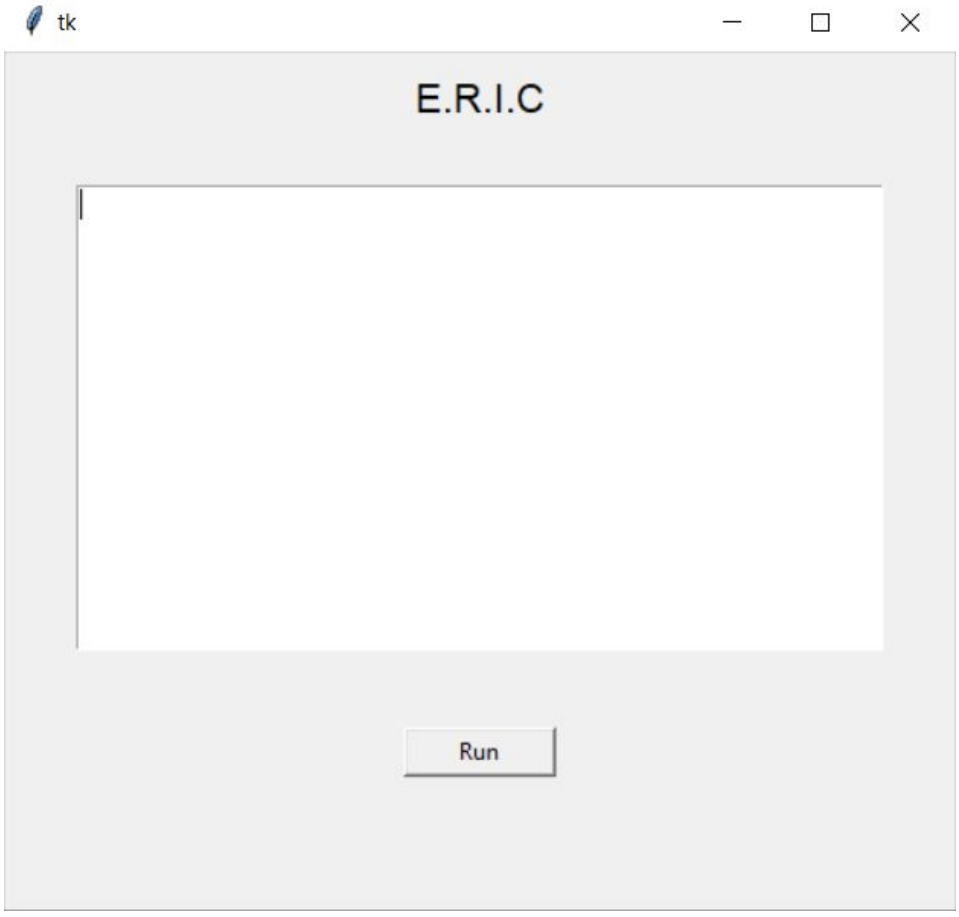
```
for 5 do
    prt "pls"
fin

Output:
pls
pls
pls
pls
pls
```

If statements

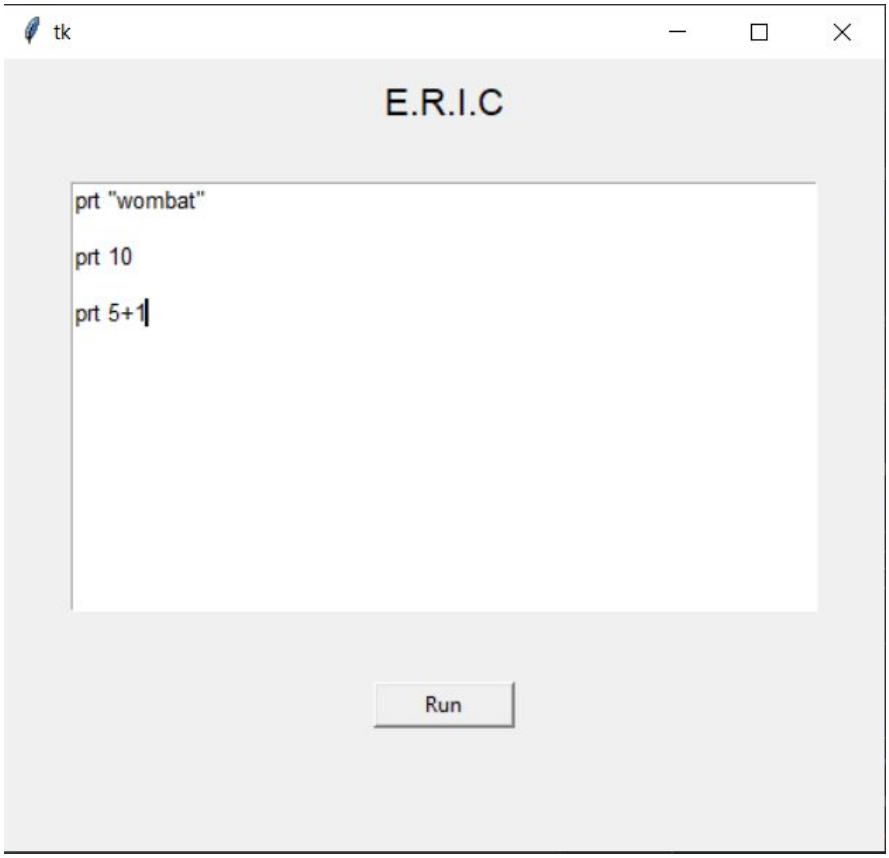
In eric the if statements syntax are almost the same as the for loop “if *condition* do” and it will also be limited to “fin” so the code that will be run is between do and fin.

```
if 10==10 do
    prt "Venti"
fin

if 10>10 do
    prt "Venti"
fin

Output:
Venti
```

Starting the Program



As the program starts it will prompt the GUI with an empty textbox and a run button.



Output:
>>> wombat
>>> 10
>>> 6

The textbox is where you can write the code for E.R.I.C programming language and after you are done with your code just type run and the output will be displayed in the command line.

Demo Link

https://drive.google.com/file/d/1zL7KH1LgbBMz_qnJ-jUNLpLLQFnaUQAS/view?usp=sharing

GIT Link

https://github.com/rfrm99/Assignments/tree/master/Semester%203/AoA_Project