Team Members:
Vikas Rajput : 201551064
Akash Sharma: 201551097
Shivvrat Arya: 201551059
Saurabh Singh Yadav: 201551084

LEARN LANGUAGE FINAL REPORT

April 13, 2018

1 Introduction

A compiler is computer software that transforms computer code written in one programming language into another programming language. Compilers are a type of translator that support digital devices, primarily computers.

We all have experienced difficulties in learning **PROGRAMMING**. Some people even stop learning since there are no languages in which you can easily learn coding. So we tried to create a language which would be easy to learn and will also contain the basics of programming that are required. It contains constructs like looping, switch case, use-statements, necessary data types only, etc. So thus it is quite clear that we didn't added complex things like pointers and the unnecessary things that comes with other programming languages.

We named our language **LEARN** since we have created it for new coders to learn programming in an easy and effective way. We divided the code so that everything can be read easily and comments are thus not required. We tried to neglect all those things which may discourage a newbie. Everyone of us start our programming journey with languages like C, C++, JAVA etc. These languages require firstly to learn how they work and to write a simple code to print something as vague as HELLO WORLD on the console we have to write quite a few lines. Then comes the syntactical errors and the horrifying error checking of different compilers which shows 100 errors even if one word is written wrong without mentioning where the error is. So these languages force us to first learn them, then programming and if a person doesn't understand the concepts of that language then they may even leave programming.

2 Why is LEARN better for newbies

- Can be used in school curriculum's for basic understanding of programming.
- We have removed all complex things from PROGRAMMING which can discourage an amateur.
- We have kept the most basic things which are the CORE part of programming.
- The syntax of LEARN is very easy to understand.
- The naming conventions are used so that the user can easily identify the constructs.

Now we will try to expand on these points in detail and will also see why we were motivated to create this language.

3 Motivation behind LEARN

3.1 Can be used in school curriculum's for basic understanding of programming

Designed with a purpose to teach the BEGINNERS. So when all of us started programming we were introduced to **C**. It was a difficult time for us learning C since the pointers and recursion were creating many errors. We were stuck how recursion worked and the errors were our enemies. We were not able to articulate our thoughts into the code and were discouraged by that. Many of us thought that programming is not for us.

But then we were introduced to **JAVA** which was slightly better language then C but it also required quite a lot lines of codes to do simple things. We thought that this is the best form of programming and it cannot get any better than this. We were doing Object Oriented Programming in JAVA which improved our level in the language. But still we required to write a bunch of lines to print things on screen and do basic things like adding and subtracting.

Then we were introduced to **PYTHON**. It was easy language and each and everything was same as english. In Java if you wanted to print you had to write System.out.println() but in python it was as easy as print(). We had to import nothing and everything as basic as printing and taking input came as it is. So we were motivated by all these ups and downs and wanted to create a language which would be easy to write and understand but would contain all the basic features of a programming language. And thus the idea to create **LEARN** began.

3.2 We have removed all complex things from PROGRAMMING which can discourage an amateur

As mentioned above we tried to keep only those things which were the basic elements of learning a programming language. We removed tough concepts like :

- Function Pointer,
- Pointer to array of pointers,
- Near, Huge and Far pointers and pointer normalization,
- Use of volatile variables to limit compiler optimization,
- Making programs thread-safe and scalable, Pointer arithmetic for Multidimensional arrays (specially while passing them to functions).

As we have all found all these concepts as magic in C when we first start coding and it is quite demotivating to loose our confidence in the language and programming due to these.
We tried to use the basic concepts of Programming like:

- Looping construct
- switch-case
- functions
- imports
- data types
- and much more.....

These concepts are the building blocks of a programming language which are present in each programming language. Thus we Incorporated these into our language so that a newbie knows everything that is required in basic coding.

3.3 The syntax of LEARN is very easy to understand

Since we all know that in programming languages like C and JAVA we had to remember a lot of code before writing even a line of code which is relevant to the work we want to do. The things like public static void main is the thing that comes to mind of a JAVA programmer first then even thinking about the conceptual part of solving the question.
We introduced **CROCODILE** braces so that the user can easily distinguish between the parts of the program. We created three parts of the program which includes :

- The use part
- The method part
- The start part

So the use part of the code contains any of the files that are required for importing. Even if there are no imports then too the user have to write each and every part.

The method part contains all the methods that the user want to define. It contains everything about the methods. Its definition, its parameters, etc. It is easy to define methods here since all the methods are defined at the same place and a new person won't be confused by this. All the methods have a basic structure.

The start part contains the most important part of the code. IT SHOULDN'T BE EMPTY. It is just like the main methods in JAVA and C.

3.4 The naming conventions are used so that the user can easily identify the constructs

So we have used the most basic naming conventions that can be used in a programming language. For example,

- The main part of the code is named as START since all our code resides in this and the compilation starts form this part only.
- The part in which we define methods is named as method and the same goes for the use part.
- The looping construct is named as loop.
- If and elif are the constructs for conditional statements
- The data types are named as integer|real|character. and many more.

Since the naming is easy to understand thus the newbie can easily write and read code written in **LEARN**. The person don't need to learn anything new if they already know basic English.

4 Features of LEARN

- All keywords are named such that they dene the meaning directly.
- We have kept only the basic features of programming. Thus it is easy to understand.
- Since we are using "<" and ">" to dene the boundaries of everything,
- it is easy to identify the code blocks.

5 Grammar

1. code → use <use_stmts> methods <method_def> start <method_def>
2. use_stmts → use_stmt; use_stmts | ϵ
3. use_stmt → **NAME**
4. method_def → **NAME** (parameters) { statements } method_def | ϵ
5. statements → statement; statements | looping_stmt statements | conditional_stmt statements | ϵ
6. parameters → parameter, parameters | parameter | ϵ
7. parameter → datatype **NAME**
8. datatype → integer | real| character
9. method_def → statement; statements | ϵ
10. statement → assignment_stmt | declaration_stmt | method_call_stmt | return_stmt
11. return_stmt → return value
12. declaration_stmt → datatype name_list
13. name_list → **NAME** = assign, name_list | **NAME**, name_list | **NAME** = assign | **NAME**
14. assign → value
15. value → **NAME** | **INTEGER_VALUE** | **REAL_VALUE**
16. assignment_stmt → **NAME** = value | array_assign_stmt | **NAME** = arithmetic_expression | **NAME** = boolean_expression | **NAME** = '**STRING**' | **NAME** = method_call_stmt
17. array_assign_stmt → **NAME**[index] = value | **NAME**[index] = **STRING**
18. index → **NAME** | **INTEGER_VALUE**
19. conditional_stmt → if_stmt | switch_stmt
20. if_stmt → if(expression) { statements } ELIF else { statements } | if(expression) { statements } ELIF
21. ELIF → elif(expression) { statements } ELIF | ϵ

22. switch_stmt → switch**(NAME)** { case_stmt default_stmt }
23. case_stmt → case **INTEGER_VALUE**: statements case_stmt | ϵ
24. default_stmt → default: statements | ϵ
25. expression → arithmetic_expression | boolean_expression
26. arithmetic_expression → term operator1 arithmetic_expression | term
27. term → factor operator2 term | factor
28. factor → (arithmetic_expression) | value
29. operator1 → **+** | **-**
30. operator2 → ***** | **/**
31. boolean_expression → boolean_expression logical_operator boolean_expression | (boolean_expression) | arithmetic_expression relational_operator arithmetic_expression | NOT boolean_expression | TRUE | FALSE
32. logical_operator → AND | OR
33. relational_operator → **<** | **>** | **<=** | **>=** | **==** | **!=**
34. looping_stmt → loop(boolean_expression) { statements }
35. method_call_stmt → **NAME** = **NAME** (Parameter_Names) | **NAME** (Parameter_Names) | **NAME** (**STRING**)
36. Parameter_Names → **NAME**, Parameter_Names | **NAME** | ϵ

6 What we have done so far

We firstly created grammar of "LEARN" and then after the first presentation we made some enhancements in the language. Then we created the lexical parser with the help of lex program. We used our generated language to create a parser by which we were able to check the syntactic credibility of our language.
Then we used yacc to make the semantic analysis of the language but we were not able to generate it as of now.

7 Acknowledgement

We are highly indebted to Indian Institute of Information Technology, Vadodara for their guidance and constant supervision as well as for providing necessary information regarding the project also for their support in completing the project. Special thanks to Dr. Ashish Phophalia for guiding and mentoring us during our project. Without his help completion of this project wouldn't have been possible.