A
Minor Project
on

"SPL - A Simple Programming Language"

Submitted to

CHHATTISGARH SWAMI VIVEKANAND TECHNICAL UNIVERSITY, BHILAI

For fulfillment of the award of the degree

Bachelor of Engineering
in

COMPUTER SCIENCE & ENGINEERING

Submitted By
Kanishka Tiwari
S Shashank
Sachin Karun
Yash Lovett

Guided By
Mr. Ramakant Ganjeshwar

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
CHOUKSEY ENGINEERING COLLEGE, BILASPUR (C.G)

Session 2021

DECLARATION BY THE CANDIDATES

I the undersigned solemnly declare that the report of the project work entitled **"SPL - A Simple Programming Language"** is based on my own work carried out during the course of my study under the guidance of **Mr. Ramakant Ganjeshwar (Asst. Prof.) Department of Computer Science & Engineering, Chouksey Engineering College, Bilaspur (CG)**.

I further declare that the statements made and conclusions drawn are an outcome of my project work.

_____

(Signature Of Student**)**
Kanishka Tiwari
Enrollment No.: BI3717

_____

(Signature Of Student**)**
S Shashank
Enrollment No.: BI3718

_____

(Signature Of Student**)**
Sachin Karun
Enrollment No.: BG1125

_____

(Signature Of Student**)**
Yash Lovett
Enrollment No.: BG1148

CERTIFICATE BY GUIDE

This is to certify that the project entitled **"SPL – A Simple Programming Language"** is a record of work carried out by **Kanishka Tiwari,** Enrollment No.: BI3717**, S Shashank** Enrollment No.: BI3718, **Sachin Karun** Enrollment No.: BG1125, **Yash Lovett** Enrollment No.: BG1148**,** bearing Under my guidance and supervision for the award of Degree of **Bachelor of Engineering, Chhattisgarh Swami Vivekananda Technical University, Bhilai (C.G.), India.**

To the best of my knowledge and belief the Project

i. Embodies the work of the candidate him/herself.

ii. Has not been submitted for the award of any degree.

iii. Fulfils the requirement of the Ordinance relating to the B.E degree of the University and,

iv. Is up to the desired standard in respect of contents and is being referred to the examiners.

_____

(Signature of the Guide)

Mr. Ramakant Ganjeshwar
(Asst. Prof. Dept of CSE)
(CEC, Bilaspur(CG))

Recommendation

The Project work as mentioned above is hereby being recommended and forwarded for examination and evaluation.

_____

(Signature of the Project Incharge)

Mr. Vivek Singh Rathore

(Asst. Prof. Dept of CSE)
(CEC, Bilaspur(CG))

_____

(Signature of the HOD with seal**)**

Mrs. Shanu K Rakesh

(HOD, Department of CSE)
(Chouksey Engineering College, Bilaspur(CG))

CERTIFICATE BY THE EXAMINERS

This is to certify that the project work entitled **"SPL – A Simple Programming Language"** submitted by **Kanishka Tiwari,** Enrollment No.: BI3717**, S Shashank,** Enrollment No.: BI3718, **Sachin Karun,** Enrollment No.: BG1125, **Yash Lovett** Enrollment No. : BG1148**,** have been completed under the guidance of **Mr. Ramakant Ganjeshwar, (Asst. Prof.)** Department of Computer Science & Engineering, Chouksey Engineering College, Bilaspur(CG) has been examined by the undersigned as a part of the examination for the award of **Bachelor of Engineering** degree in **Computer Science & Engineering** branch in Chouksey Engineering College, Bilaspur of **Chhattisgarh Swami Vivekanand Technical University, Bhilai(CG).**

"Project Examined and Approved"

Internal Examiner
Date:

External Examiner
Date:

_____

(Signature of the Principal with seal)

Principal
Chouksey Engineering College, Bilaspur(C.G)

ACKNOWLEDGEMENT

At every outset I express my gratitude to almighty lord for showering his grace and blessing upon me to complete this project.

Although my name appears on the cover of this project, many people had contributed in some form or the other form to this project development. I could not done this project without the assistance or support of each of the following we thank you all.

I wish to place on my record my deep sense of gratitude to my project guide, **Mr. Ramakant Ganjeshwar**, **(Asst. Prof.)** Dept. of C.S.E, C.E.C Bilaspur and my project in charge, **Mr. Vivek Singh Rathore, (Asst. Prof.)** Dept. of CSE, CEC Bilaspur for their constant motivation and valuable help through the project work. Express my gratitude to **Mrs. Shanu K Rakesh, H.O.D**. of C.S.E, C.E.C Bilaspur for his valuable suggestions and advices throughout the course. I also extend our thanks to other faculties for their cooperation during my course.

Finally I would like to thank my friends for their cooperation to complete this project.

_____ _____

(Signature Of Student**)** **(**Signature Of Student**)**

Kanishka Tiwari **S Shashank**

Enrollment No.: BI3717 Enrollment No.: BI3718

_____ _____

(Signature Of Student**)** **(**Signature Of Student**)**

Sachin Karun **Yash Lovett**

Enrollment No.: BG1125 Enrollment No.: BG1148

ABSTRACT

Since the last three decades, we have seen a rise in problem-solving through computer programming. A programming language is often chosen based on the requirements of the problem. Many existing programming languages are utilized already in different areas such as Data Science or Application Development.

This project proposes to provide a means of solving simple programming problems and help students to develop logic in a more abstract sense. The objective of this project is to design a new simple programming language that resembles pseudocode, which helps a student to focus more on the logic rather than syntactical issues of the programming language.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

Programming can be a very creative and rewarding activity. Programs can be written for many reasons, ranging from solving simple arithmetic problems to difficult data analysis problems. The past three decades have seen a rise in computer programming and several programming languages have come into existence depending on the problem being solved. Popular languages like Java, C#, etc. are widely used in application development, whereas data science problems are solved using either R or Python.

However, very few programming languages are designed purely for academic purposes. The principal motivation of this project comes from Scratch, a block-based visual programming language aimed at teaching programming to young children.

On similar lines, our project, Simple Programming Language (SPL) aims to teach programming to students in an abstract sense, similar to pseudocode, before they learn a more specialized programming language.

This project is highly motivated by our interest in the study of compiler design. It is an attempt to implement a simple programming language by implementing the different phases of a compiler and defining a grammar for the programming language.

CHAPTER 2

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

The idea of SPL mainly derives from [1] Scratch, which is a visual programming environment that allows users (primarily ages 8 to 16) to learn computer programming while working on animated stories.

The definition of the formal grammar for SPL is derived from [2] BASIC (Beginner's All-Purpose Symbolic Instruction Code). It is a family of general-purpose, high-level programming languages whose design philosophy emphasizes ease of use.

Two principal tasks involved in designing the compiler, Lexical Analysis and Parsing have been achieved based on [3] and [4], both standard texts on compiler design.

CHAPTER 3

PROBLEM DEFINITION

CHAPTER 3
PROBLEM DEFINITION

3.1 PROBLEM STATEMENT

The problem statement of this project is to define formal grammar and design a compiler for parsing this grammar to enable students to write programs.

The major reason behind choosing the project is to facilitate a means of problem-solving in an abstract manner by focusing more on the logic behind the problem being solved.

3.2 OBJECTIVE

Often students learn a programming language but fail to develop the mindset to apply logic for a problem. This project provides a means to focus more on the logic behind the problem similar to writing pseudocodes.

The principal objective is to provide a new simple programming language that abstracts out the syntactical issues to a maximum extent and yet provides all basic features required for the programming language.

CHAPTER 4

PROPOSED SYSTEM

CHAPTER 4

PROPOSED SYSTEM

4.1 FORMAL GRAMMAR

Formal grammar is a set of rules that are used to identify correct or incorrect strings of tokens in a language. The formal grammar is represented as G. It is used to generate all possible strings over the alphabet that is syntactically correct in the language. Formal grammar is used mostly in the syntactic analysis phase (parsing) particularly during the compilation.

The formal grammar of SPL has been defined as below:

```
statements  : NEWLINE* statement (NEWLINE+ statement)* NEWLINE*

statement               : KEYWORD:RETURN expr?
                        : KEYWORD:CONTINUE
                        : KEYWORD:BREAK
                        : expr

expr        : KEYWORD:VAR IDENTIFIER EQ expr
            : comp-expr ((KEYWORD:AND|KEYWORD:OR) comp-expr)*

comp-expr   : NOT comp-expr
            : arith-expr ((EE|LT|GT|LTE|GTE) arith-expr)*

arith-expr  :   term ((PLUS|MINUS) term)*

term        : factor ((MUL|DIV) factor)*

factor      : (PLUS|MINUS) factor
            : power

power       : call (POW factor)*

call        : atom (LPAREN (expr (COMMA expr)*)? RPAREN)?

atom        : INT|FLOAT|STRING|IDENTIFIER
            : LPAREN expr RPAREN
            : list-expr
            : if-expr
            : for-expr
            : while-expr
            : func-def
```

```
list-expr   : LSQUARE (expr (COMMA expr)*)? RSQUARE

if-expr     : KEYWORD:IF expr KEYWORD:THEN
              (statement if-expr-b|if-expr-c?)
            | (NEWLINE statements KEYWORD:END|if-expr-b|if-expr-c)

if-expr-b   : KEYWORD:ELIF expr KEYWORD:THEN
              (statement if-expr-b|if-expr-c?)
            | (NEWLINE statements KEYWORD:END|if-expr-b|if-expr-c)

if-expr-c   : KEYWORD:ELSE
              statement
            | (NEWLINE statements KEYWORD:END)

for-expr    : KEYWORD:FOR IDENTIFIER EQ expr KEYWORD:TO expr
              (KEYWORD:STEP expr)? KEYWORD:THEN
              statement
            | (NEWLINE statements KEYWORD:END)

while-expr  : KEYWORD:WHILE expr KEYWORD:THEN
              statement
            | (NEWLINE statements KEYWORD:END)


func-def    : KEYWORD:FUN IDENTIFIER?
              LPAREN (IDENTIFIER (COMMA IDENTIFIER)*)? RPAREN
              (ARROW expr)
            | (NEWLINE statements KEYWORD:END)
```

4.2 COMPILER AND INTERPRETER

Compilers and interpreters are programs that help convert the high-level language into machine codes to be understood by the computers. SPL supports both an interpreter and a compiler for executing the code.

The different components of a compiler such as the lexical analyzer, parser, symbol tables, and the interpreter have been implemented on top of the Python programming language.

The user can execute instructions one by one through the shell and as well as write a script file and compiler it to produce the output.

4.3 FEATURES OF SPL

1. SPL is a dynamically typed programming language with the support of primitive data types such as Integer, Float, String, and Boolean.

2. SPL provides support for arithmetic operations through binary operators.

3. SPL supports comparison operators for decision-making constructs. It defines three decision-making constructs, IF-THEN, IF-THEN-ELSE, and IF-ELIF-ELSE.

4. SPL provides two ways to perform iterations, through FOR and WHILE constructs.
5. Programs in SPL can be typed functionally using the support for user-defined functions.
6. SPL provides a basic data structure LIST equivalent to the classical array.

4.4 COMPONENTS OF SPL

The project consists of 4 components:

1. Lexical Analyzer: defined the various tokens of the language
2. Interpreter: to execute instructions line by line through the shell
3. Parser: defines grammar rules and checks for errors in the program
4. Symbol Table: defines maintains information of program data

These different components are included under the file **main.py**

The shell is invoked through the file **shell.py**

CHAPTER 5

SYNTAX DEFINITIONS

CHAPTER 5

SYNTAX DEFINITIONS

5.1 VARIABLES AND OPERATORS

SPL is a dynamically typed programming language. A variable is not needed to be assigned a data type explicitly. Variables can directly be assigned values and based on this; the compiler assigns the data type to the variable in the symbol table.

SPL supports different arithmetic and comparison operators to perform simple computations.

The syntax for a variable is defined as below:

```
VAR var_name = value
```

The variables may contain alphabets, digits, and underscore. The name must start either with an alphabet or an underscore only. It must not contain any special characters other than underscore.

The different operators supported in SPL are as below:

^	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction
=	Assignment
==	Equals To
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To

Table 5.1 Operators in SPL

The addition operator, '+', acts as concatenation in the case of a String operand.

5.2 CONDITIONAL STATEMENTS

SPL provides support for conditional execution through three different conditional constructs.

1. IF-THEN

2. IF-THEN-ELSE

3. IF-ELIF-THEN-ELSE

The syntax of these constructs is defined as below:

1. IF-THEN

```
IF comp_expr THEN expr
```

2. IF-THEN-ELSE

```
IF comp_expr THEN expr ELSE expr
```

3. IF-ELIF-THEN-ELSE

```
IF comp_expr THEN expr ELIF comp_expr THEN expr ELSE expr
```

5.3 ITERATIVE EXECUTION

SPL supports two looping constructs, FOR and WHILE. The syntax is defined as below.

1. FOR LOOP

```
FOR id = expr TO expr (STEP expr) THEN
    ...
END
```

2. WHILE LOOP

```
WHILE cond THEN
    ...
END
```

5.4 FUNCTIONS

Functions are reusable pieces of code that are written once in a program and can be later executed multiple times. SPL supports the definition of user-defined functions in the program.

The syntax of a function is defined as,

```
FUN function_name([params]) -> expr
```

The function can also have multiple expressions in the body. This is defined as,

```
FUN function_name([params])
    ...
END
```

To call a function at any point in the program, we write

```
function_name([params])
```

5.5 LISTS

The list is a container for storing multiple data values under a single name. It is similar to the array data structure. However, a list can contain data elements of different data types. It supports basic built-in functions such as LEN, APPEND, etc. to operate on the lists.

In SPL, list is defined as below.

```
VAR list_name = [(val1,val2,...)]
```

5.6 COMPILATION

SPL provides the shell for executing instructions sequentially. It also provides a compiler to compile and execute script files.

The code file can be written in a file with an extension ".spl" and it can be executed using the RUN command in the shell.

The syntax of the RUN command is as below:

```
RUN("file_name.spl")
```

CHAPTER 6

SNAPSHOTS AND EXECUTION

CHAPTER 6

SNAPSHOTS AND EXECUTION

SHELL

```
SPL 1.0 2022-02-01 16:17:37.729541
 on  Windows
prompt >
```

Fig 6.1. SHELL

VARIABLES AND ARITHMETIC OPERATORS

```
SPL 1.0 2022-02-01 16:17:37.729541
 on  Windows
prompt > VAR X = 3
3
prompt > VAR Y = 8
8
prompt > X+Y
11
prompt > X*Y
24
prompt > Y^X
512
prompt > Y/X
2.6666666666666665
prompt > X-Y
-5
prompt >
```

Fig. 6.2 Variables and arithmetic operators

CONDITIONAL EXECUTION

```
prompt > IF Y>X THEN VAR Z = X+Y ELIF X<Y THEN VAR Z = X*Y ELSE VAR Z = X-Y
11
prompt > Z
11
prompt >
```

Fig. 6.3 Conditional Execution

ITERATIVE EXECUTION

```
prompt > FOR X = 1 TO 20 STEP 2 THEN X+2
[3, 5, 7, 9, 11, 13, 15, 17, 19, 21]

prompt > WHILE X > 0 THEN VAR X = X-2
[13, 11, 9, 7, 5, 3, 1, -1]
```

Fig. 6.4 Iterative Execution

FUNCTIONS

```
prompt > FUN square(X) -> X^2
<function square>
prompt > square(3)
9
```

Fig. 6.5 Functions

LISTS

```
prompt > VAR L = [1,2,3,4]
[1, 2, 3, 4]
prompt > APPEND(L,5)
0
prompt > L
[1, 2, 3, 4, 5]
prompt >
```

Fig. 6.6 Lists

SCRIPT FILE EXECUTION

Execution of a script file involves two steps:

1. Write the code in a text editor and save it with the extension: **.spl**
2. Execute this file with the RUN command from the SPL interpreter.

The file **sumoflist.spl** computes sum of elements in a list

```
FUN sumoflist(list)
        VAR sum = 0

        FOR i = 0 TO LEN(list) THEN
                VAR sum = sum + list/i
        END

        RETURN sum
END

PRINT("HELLO WORLD")

VAR X = sumoflist([1,2,3,4,5])
PRINT(X)
```

```
prompt > RUN("sumoflist.spl")
HELLO WORLD
15
0
```

Fig. 6.7 Script File Execution

CHAPTER 7

CONCLUSION AND FUTURE WORK

CHAPTER 7

CONCLUSION AND SCOPE OF FUTURE WORK

The proposed programming language incorporates features necessary for problem-solving at a beginner level. It supports constructs like variables, operators, conditionals, iterations, functions, and lists.

In a later version of the project, the features of interest that would be incorporated into it are user input, exception handling, object-oriented implementations, and libraries. These features will be useful for the user to perform complex problem solving and provide an extensive implementation of higher data structures.

REFERENCES

[1] Maloney, John, et al. "The scratch programming language and environment." *ACM Transactions on Computing Education (TOCE)* 10.4 (2010): 1-15.

[2] BASIC, TI-BASIC. "Beginner's All-purpose Symbolic Instruction Code."

[3] Holub, Allen I. *Compiler design in C*. Vol. 5. Englewood Cliffs, NJ: Prentice Hall, 1990.

[4] Aho, Alfred V., et al. *Compilers: Principles, Techniques, & Tools*. Pearson Education India, 2007.

[5] Severance, Charles R. "Python for everybody: exploring data in python 3." (2016).

[6] Mitchell, John C., and Krzysztof Apt. *Concepts in programming languages*. Cambridge University Press, 2003.