
# COMPILER AND VIRTUAL MACHINE FOR A PROGRAMMING LANGUAGE

## SER 502 – TEAM 12

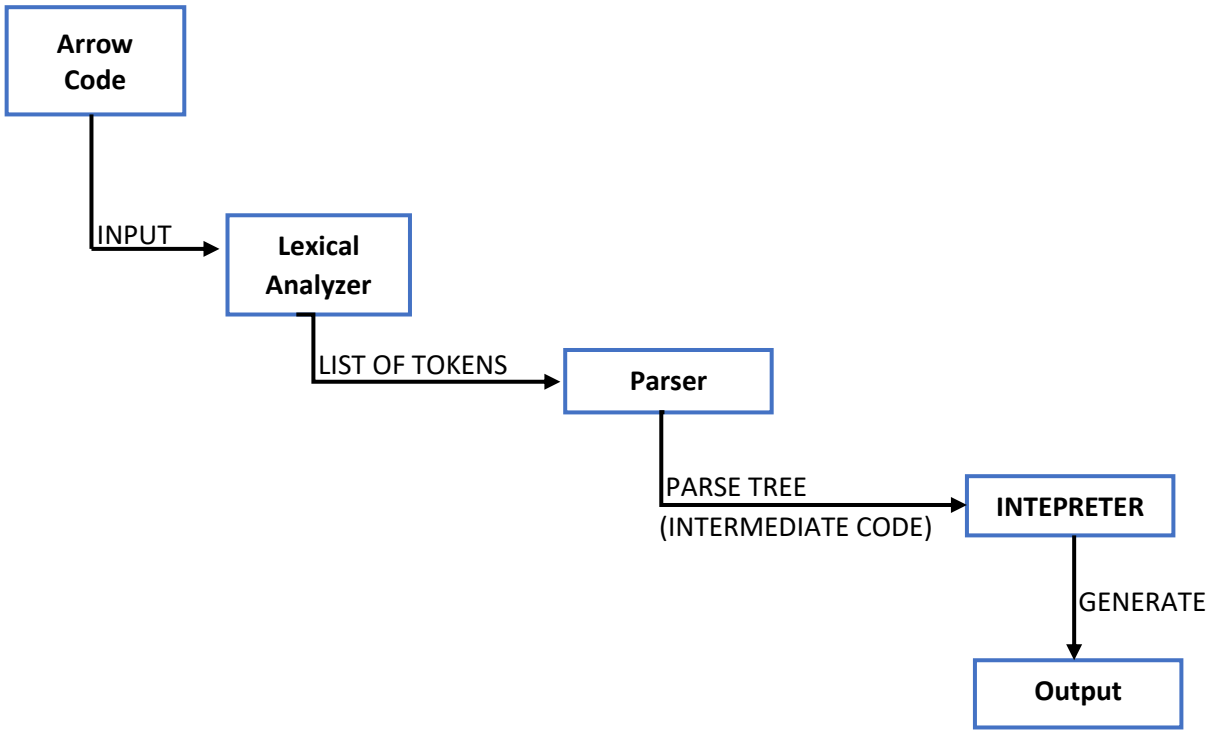
## SPRING 2018


1. Harshitha Katpally hkatpall@asu.edu
2. Divya Yadamreddi dyadamre@asu.edu
3. Venkata Akhil Madaraju vmadaraj@asu.edu
4. Venkata Sai Shirisha Kakarla vkakarla@asu.edu

**LANGUAGE NAME**
Arrow Programming Language
File Extension: *.arr

**PROGRAMMING PARADIGM**
Imperative

**DESIGN**




**Design Decisions:**
We decided to design and program our own lexical analyzer, parser and a runtime environment using the declarative style of programming in prolog because this style is well-suited for implementing language grammars using definite clause grammar and it allows to do a lot of work in less lines of code.

**Lexical Analysis and Parsing Technique:**
Parsing begins with lexical analysis. Lexical analysis will be done in prolog. The lexical analyzer will read in the code from the user. Once the code is read in, it is broken down into tokens based upon the grammar rules set. Once we have the tokens, these tokens are passed through the parser that is a prolog code with the grammar defined in it using the prolog predicates for each grammar rule, to generate the syntax tree/parse tree.

**Intermediate Code:**
The generated parse tree will be the intermediate code for our programming language.

**Interpreter Design:**
The runtime/Interpreter is implemented by traversing node by node through the parse tree. Each node denotes an evaluation step that needs to be performed. The environment at each node is looked up or updated via a list data structure in prolog.The runtime will be programmed from scratch using prolog.

**TOOLS USED**
We have decided to use prolog, declarative style of programming to design our Lexical analyzer, Syntactic analyzer and create the runtime environment for Arrow Programming language.

**RECURSION METHOD FOR GRAMMAR**
We have written the grammar using tail recursion and used the BNF notation to represent our grammar rules.

**DATA TYPES**
ARROW will support 4 data types - Integer, Float, Bool, String. The domains for these data types are as given below:

- INT: Supports all positive numbers.
- FLOAT: Supports all floating point numbers.
- BOOL: Supports Boolean Values, 'True' or 'False'.
- STRING: Supports any string excluding a new line.

- To declare a variable of a particular data type a particular syntax should be followed.
Ex. - INT integer; - This statement declares a variable 'integer' of type INT.
- The identifiers used to declare variables should start with an alphabet and can contain numbers.
- Data types can be declared and initialized at the same time.
Ex. - INT integer = 9;

**OPERATORS**
- The language will support basic arithmetic operations i.e. addition, subtraction, multiplication, division denoted by '+', '-', '*' and '/' respectively.
- It also supports the boolean operators AND, OR and NOT represented as 'and', 'or' and 'not' respectively. These boolean operators are defined to be used in conditional statements, to check if both conditions are true, either of the conditions is true and a condition is not true.
- Finally, the language supports the comparison operators, '<', '>', '<=', '>=' and '=='. These operators can be used to compare two expressions.

**CONDITIONALS**
- ARROW supports the 'if-else' conditional construct which can be used to evaluate decision making situations. If the condition written after the 'if' keyword evaluates to true, then the block of code under if is executed, otherwise the block of code under else is executed. The 'if-else' conditional can be used without the 'else' part i.e. it can have only an 'if' part.



**LOOPS**

The language supports the ‘while’ looping constructs, which loops through a block of statements as long as the condition is true. The ‘while’ loop works in a way that it first checks if the condition is true and only if it is true the block of statements within the braces is evaluated. Once the condition is false, the control exits the loop.

**COMMENTS**

The comments in ARROW must be enclosed within a pair of symbols i.e. ‘<<’ and ‘>>’. This supports only single line commenting and whatever is written within this set of symbols is never executed and is only required to make the code understandable.

**TERMINATORS**

-Arrow has BEGIN and END as keywords being used to indicate the start and end of a piece of code in our programming language.

-Arrow supports semi-colon (‘;’) and next line (new line) as line terminators to improve readability of programming language.

-White space used in ARROW is not only to increase the readability of the program but also to separate tokens from each other.

**PRINT**

-To print any string to the console ARROW supports a print functionality where ‘print’ keyword is followed by a space and a string to be printed.

## GRAMMAR

**Data Types:**

INT := /^[0-9]+$/
FLOAT := /^[0-9]+\.[0-9]+$/
BOOL := /‘True’ | ‘False’/
STRING := /‘\”’(.)*?’\”’/
DATATYPE := INT | FLOAT | BOOL | STRING

**Identifier:**

IDENTIFIER := ^[a-z]+[a-z0-9]*$

**Operators:**

ADD := ‘+’
SUB := ‘-’
MUL := ‘*’
DIV := ‘/’
AND := ‘AND’ | ‘and’
OR := ‘OR’ | ‘or’
NOT := ‘NOT’ | ‘not’
ASSIGN := ‘=’
COMPARE := ‘>’ | ‘<’ | ‘<=’ | ‘>=’ | ‘==’
CONDOP := AND | OR | NOT

**Conditional Statements Keywords:**

IF := ‘if’
ELSE := ‘else’

**Loops Keywords:**

WHILE := ‘while’

**Comments:**

SCOMMENT := ‘<<’
ECOMMENT := ‘>>’

**Blocks:**

SBLOCK := ‘{‘
EBLOCK := ‘}’

**Terminators:**

DELIMITER := ‘;’
SEPARATOR := ‘,’
SPACE := / /
NEXTLN := /[\n]+/
TAB := /[\t]+/
BEGIN := ‘begin’
END := ‘end’

**Parenthesis:**

OPARENTHESIS := ‘(‘
CPARENTHESIS := ‘)’

**Print:**

PRINT := ‘print’

**RULES**

program := BEGIN NEXTLN block NEXTLN END.

block := comment NEXTLN declaration | declaration | declaration statements.

declaration := DATATYPE SPACE IDENTIFIER DELIMITER NEXTLN | DATATYPE SPACE IDENTIFIER ASSIGN assignment DELIMITER NEXTLN

assignment := expression | BOOL | STRING

statements := IDENTIFIER ASSIGN assignment DELIMITER NEXTLN | ifelse NEXTLN | while NEXTLN | print NEXTLN | expression DELIMITER NEXTLN statements | ifelse NEXTLN statements | while NEXTLN statements | print NEXTLN statements

print := PRINT SPACE STRING

ifelse := IF '(' condition ')' SBLOCK statements EBLOCK | IF '(' condition ')' SBLOCK statements EBLOCK ELSE SBLOCK statements EBLOCK

while := WHILE '(' condition ')' SBLOCK statements EBLOCK

condition := IDENTIFIER SPACE COMPARE SPACE expression | IDENTIFIER SPACE COMPARE expression CONDOP condition | BOOL

comment := SCOMMENT STRING ECOMMENT

expression := term ADD expression | term SUB expression | term

term := factor MUL term | factor DIV term | factor

factor := OPARENTHESIS expression CPARENTHESIS | INT | FLOAT | IDENTIFIER