SER 502 Project
DevLingo
Team - 20

Manmeet Singh
Jay Sachin Chiddarwar
Ajinkya Abhinay Pise
Debesh K Pradhan
Rahul Balulmath

Introduction

Language Name – DevLingo

Systems Used – MacOS

Tools Used –

- ANTLR
- IntelliJ Idea

Extensions used –

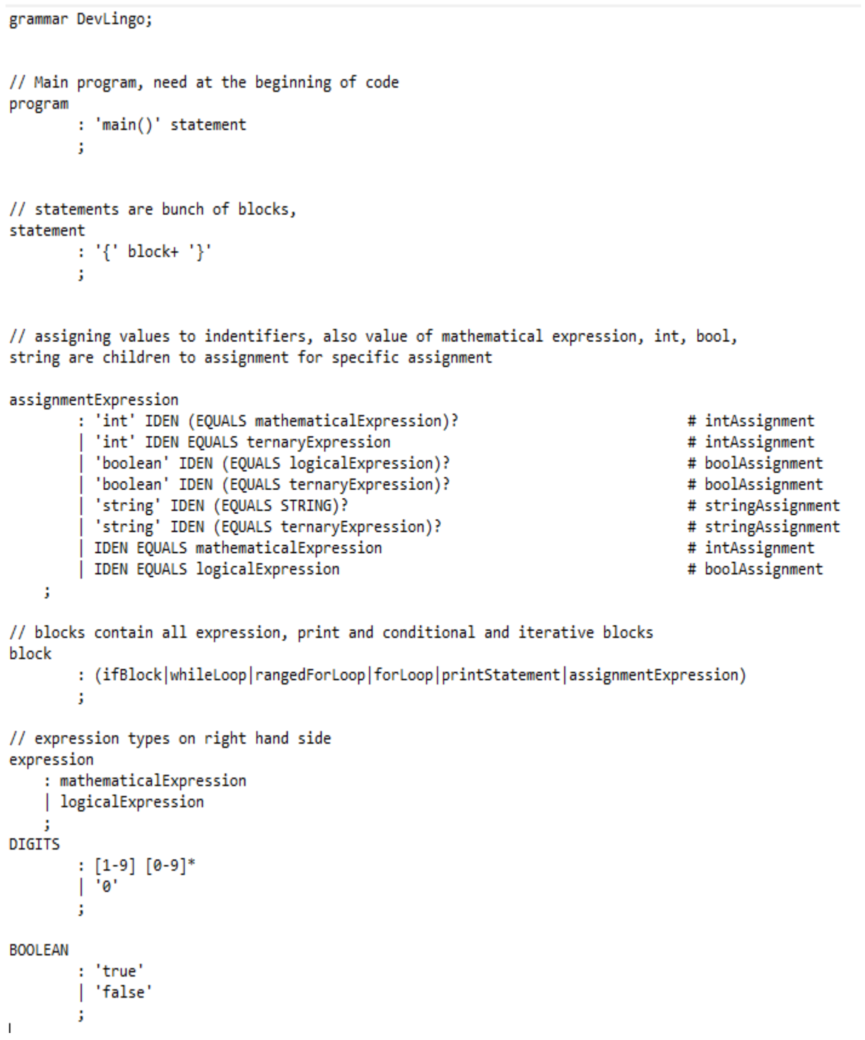
- .dvlg (Input Program)
- .dvac(Intermediate Code)

Basic Features

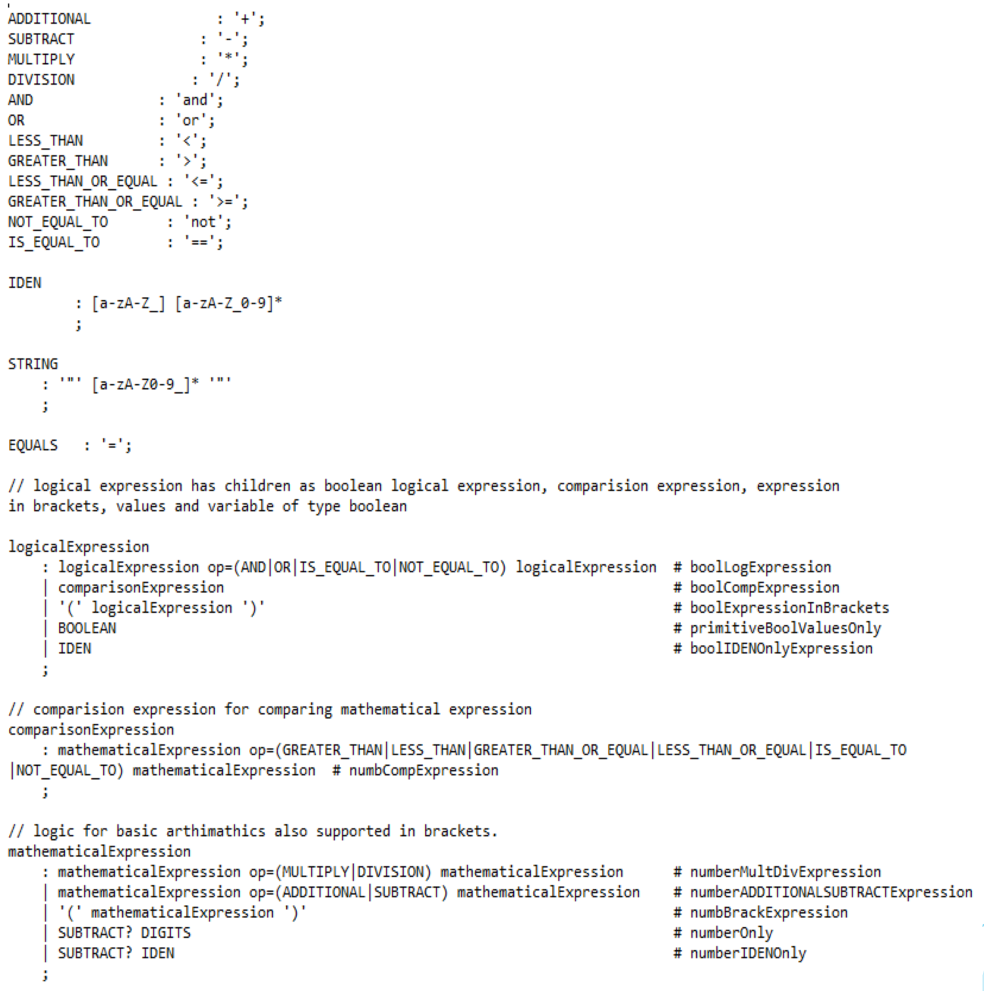
1. **Boolean values**: “**Boolean**” support Boolean operators: The language supports Boolean operators such as “**and**”, “**or**”, and “**not**” for working with Boolean values.
2. **Numeric operators**: The language supports “**int**” as the numeric operator. It supports all the basic arithmetic computations such as **addition, multiplication, subtraction, and division**.
3. **String value assignments**: The language supports string value assignments as “**string**”. Basic operations such as printing and assigning to an identifier are implemented.
4. **Assignment operator**: The language includes an assignment operator which helps to assign a value to an identifier.
5. **Conditional statements**: Traditional **if-else** statements and **ternary operators** have been included in the language for conditional statements.
6. **Iterative loops**: The language supports iterative loops such as **for loop, while loop, and for-range loops** for executing a block of code repeatedly.
7. **Print statement**: The language includes a print statement “**printf**” for printing out the values of identifiers. This supports all the data types covered in the language.

Grammar

Grammar rules are present in .g4 file



```
grammar DevLingo;


// Main program, need at the beginning of code
program
        : 'main()' statement
        ;


// statements are bunch of blocks,
statement
        : '{' block+ '}'
        ;


// assigning values to indentifiers, also value of mathematical expression, int, bool,
string are children to assignment for specific assignment

assignmentExpression
        : 'int' IDEN (EQUALS mathematicalExpression)?                          # intAssignment
        | 'int' IDEN EQUALS ternaryExpression                                 # intAssignment
        | 'boolean' IDEN (EQUALS logicalExpression)?                          # boolAssignment
        | 'boolean' IDEN (EQUALS ternaryExpression)?                          # boolAssignment
        | 'string' IDEN (EQUALS STRING)?                                      # stringAssignment
        | 'string' IDEN (EQUALS ternaryExpression)?                           # stringAssignment
        | IDEN EQUALS mathematicalExpression                                  # intAssignment
        | IDEN EQUALS logicalExpression                                       # boolAssignment
    ;

// blocks contain all expression, print and conditional and iterative blocks
block
        : (ifBlock|whileLoop|rangedForLoop|forLoop|printStatement|assignmentExpression)
        ;

// expression types on right hand side
expression
    : mathematicalExpression
    | logicalExpression
    ;
DIGITS
        : [1-9] [0-9]*
        | '0'
        ;

BOOLEAN
        : 'true'
        | 'false'
        ;
```



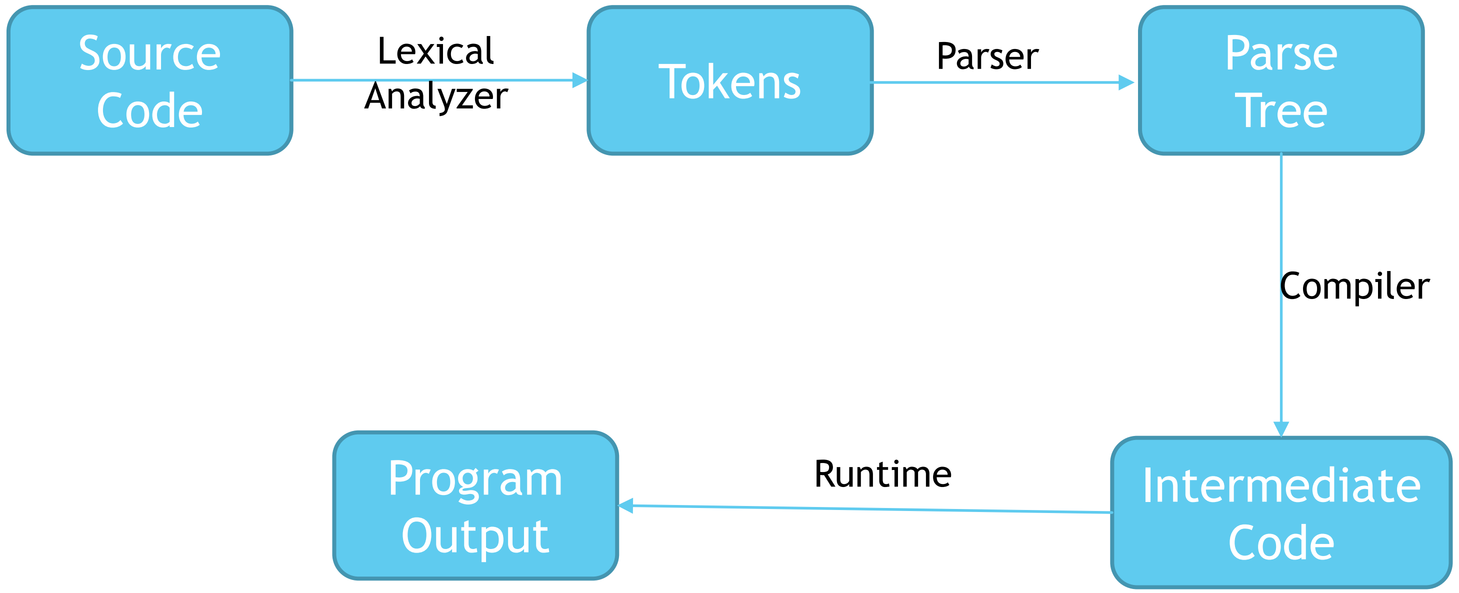
```
ADDITIONAL              : '+';
SUBTRACT              : '-';
MULTIPLY              : '*';
DIVISION             : '/';
AND                  : 'and';
OR                   : 'or';
LESS_THAN            : '<';
GREATER_THAN         : '>';
LESS_THAN_OR_EQUAL : '<=';
GREATER_THAN_OR_EQUAL : '>=';
NOT_EQUAL_TO        : 'not';
IS_EQUAL_TO         : '==';

IDEN
        : [a-zA-Z_] [a-zA-Z_0-9]*
        ;

STRING
    : '"' [a-zA-Z0-9_]* '"'
    ;

EQUALS   : '=';

// logical expression has children as boolean logical expression, comparision expression, expression
in brackets, values and variable of type boolean

logicalExpression
    : logicalExpression op=(AND|OR|IS_EQUAL_TO|NOT_EQUAL_TO) logicalExpression  # boolLogExpression
    | comparisonExpression                                                    # boolCompExpression
    | '(' logicalExpression ')'                                               # boolExpressionInBrackets
    | BOOLEAN                                                                 # primitiveBoolValuesOnly
    | IDEN                                                                    # boolIDENOnlyExpression
    ;

// comparision expression for comparing mathematical expression
comparisonExpression
    : mathematicalExpression op=(GREATER_THAN|LESS_THAN|GREATER_THAN_OR_EQUAL|LESS_THAN_OR_EQUAL|IS_EQUAL_TO
|NOT_EQUAL_TO) mathematicalExpression  # numbCompExpression
    ;

// logic for basic arthimathics also supported in brackets.
mathematicalExpression
    : mathematicalExpression op=(MULTIPLY|DIVISION) mathematicalExpression      # numberMultDivExpression
    | mathematicalExpression op=(ADDITIONAL|SUBTRACT) mathematicalExpression    # numberADDITIONALSUBTRACTExpression
    | '(' mathematicalExpression ')'                                            # numbBrackExpression
    | SUBTRACT? DIGITS                                                          # numberOnly
    | SUBTRACT? IDEN                                                            # numberIDENOnly
    ;
```

```
// basic logical expression for conditional and iterative block
conditionalExpression
    : '(' logicalExpression ')'
    ;


// contitional block and with zero or many else if and zero or one else block
ifBlock
    : 'if' conditionalExpression statement (else_ifBlock)* (else_expr)?
    ;


else_ifBlock
    : 'else if' conditionalExpression statement
    ;


else_expr
    : 'else' statement
    ;


// iterative based on condition begin true
whileLoop
    : 'while' conditionalExpression statement
    ;


// iteration based on from to val one more than first argument and till second argument
rangedForLoop
    : 'for' IDEN 'in' 'range' '(' rangeVal ';' rangeVal ')' statement
    ;

rangeVal
        : IDEN
        | DIGITS
        ;
```

```
forLoop
    : 'for' '(' assignmentExpression ';' logicalExpression ';' variable_change_part ')' statement
    ;

// supported for variable change part in for looop
variable_change_part : increment_expression
                     | decrement_expression
                     ;

decrement_expression : IDEN '--'
                     | '--' IDEN;


increment_expression : IDEN '++'
                     | '++' IDEN;


ternaryExpression
    : conditionalExpression '?' expression ':' expression
    | conditionalExpression '?' BOOLEAN ':' BOOLEAN
    | conditionalExpression '?' STRING ':' STRING
    ;
printStatement
    : 'printf' '(' (DIGITS|BOOLEAN|IDEN|mathematicalExpression|logicalExpression|STRING) ')'
    | 'printf' '(' STRING ',' (IDEN|BOOLEAN|STRING|DIGITS) ')'
    ;


WHITE_SPACES    : [ \t\r\n]+ -> skip;
```

Program Flow

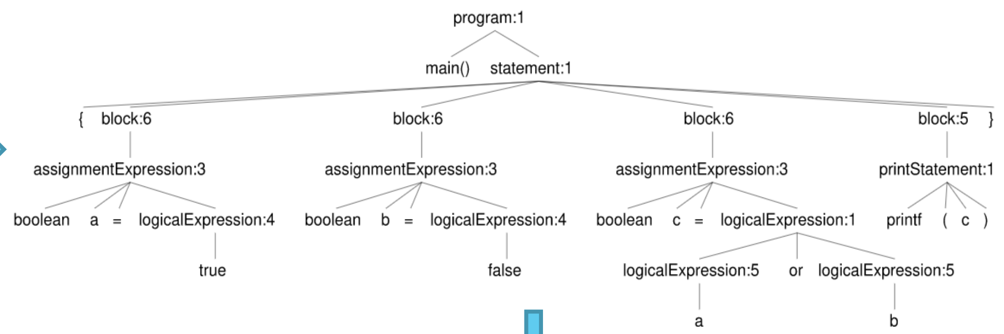
Flow of Program



Flow of a Sample Program

```
main(){
    boolean a = true
    boolean b = false

    boolean c = a or b
    printf(c)
}
```

Source Code



Parse Tree

```
LOAD ACC true
LOAD a ACC
LOAD ACC false
LOAD b ACC
LOAD ACC a
LOAD B ACC
LOAD ACC b
LOAD C ACC
OR ACC B C
LOAD c ACC
PRINT c
```

Intermediate code

```
Output

true

```

Output

Sample Runs

Sample Run – 1

Code

```
main(){
    int a = 5
    if(a > 10){
        printf("Success")
    }else{
        printf("Fail")
    }
}
```

Output

```
Output

Fail

```

Sample Run - 2

Code

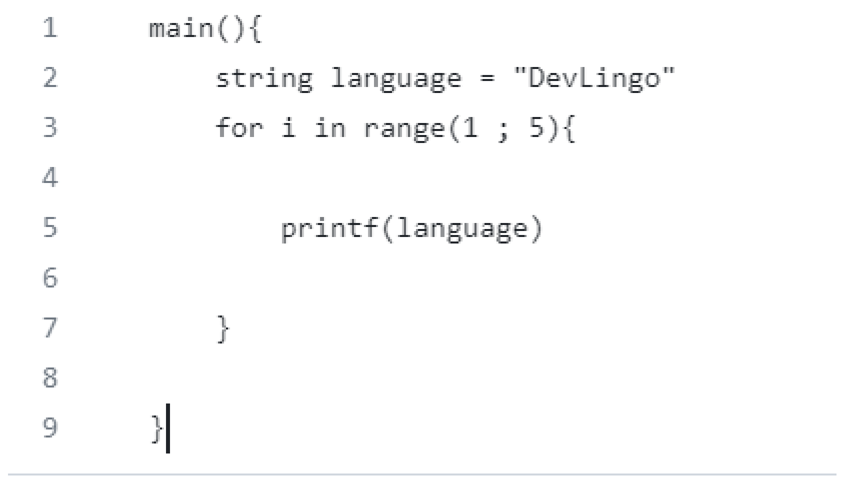
```
main(){
    boolean a = true
    boolean b = false

    boolean result1 = a and b
    printf(result1)

    boolean result2 = a or b
    printf(result2)


}
```

Output

```
Output

false
true

```

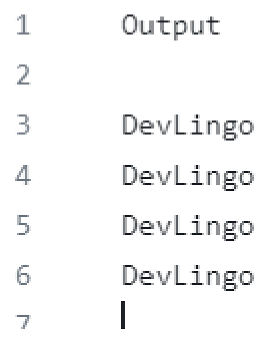
Sample Run - 3

Code



```
main(){
    string language = "DevLingo"
    for i in range(1 ; 5){

        printf(language)

    }

}
```

Output



```
Output

DevLingo
DevLingo
DevLingo
DevLingo

```

Sample Run - 4

Code

```
main(){

    string check = "Hello"
    int a = 0
    while(a < 4){
        printf(check)
        a = a+1;
    }


}
```

Output

```
Output

Hello
Hello
Hello
Hello

```

Sample Run - 5

Code

```
main(){
    for(int i=0 ; i<5; i++){
        printf(i)
    }
}
```

Output

```
Output

0
1
2
3
4

```

Sample Run - 6

Code

```
main(){
    int a = 5;
    boolean check = a>10 ? true : false;
    printf(check)
}
```

Output

```
Output

false

```

Future Work

1. We can consider introducing new language structures or syntax to boost developer productivity and make the language more expressive. For example, we may include lambdas, functional programming structures, and enhanced concurrency support.
2. Improve performance: Improving language performance is always a worthwhile subject of study. We may investigate ways to improve the language quicker, either by enhancing the language runtime implementation or by making the language itself more efficient.
3. Extend platform support: While Java is extensively used, it may not be as well supported in other regions. Consider adding support for new platforms or settings, such as mobile or web development, to your language.
4. Improve tooling: Creating effective tools and IDEs may significantly increase developer productivity. Consider creating new tools or improving current ones to improve code completion, debugging, profiling, and other features.
5. Building a strong developer community around your language can assist assure its continued growth and development. Consider strategies to increase acceptance, produce developer documentation and tools, and assist individuals who wish to contribute to the project.

Thank You