CS 315 – Programming Languages
PROJECT 1 – Team 39

Akame Language

Ahmet Işık 21702226 Section-3

Ahmet Kaan Uğuralp 21803844 Section-1

Mehmet Yaylacı 21802347 Section-3

Instructor: Halil Altay Güvenir
Teaching Assistant(s): Duygu Durmuş, Alper Şahıstan, Irmak Türköz

Name of the Language: Akame

The BNF Description of Akame Language

1) Program Definition

Grammar G (V, T, P, S)

V(non-terminals) = {<stmt>, <assignment_stmt>, <if_stmt>, <for_stmt>, <while_stmt>, <decl_stmt>, ... }

T(terminals) = {";", "int", "float", "char", "," , "(", ")", "if", "else", ... }

P(productions) = { "<ident-list> ::= <ident> | <ident> , <ident-list>", ... }

S start variable, S is a member of V, <program>

2) Types and Constants

<char> : a | b | c | d ... | A | B | C ... | _ | $

<digit>: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<sign>: + | -

<ident> : <char> | <ident> <char> | <ident> <digit>

<func_ident> : <char> | <ident> <char> | <ident> <digit>

<int_const>: <sign> <digit> | <int_const> <digit>

<ident_list> : <ident> | <ident> , <ident_list>

<true> : true

<false> : false

<idc> ::= <id>

| <const>

| <expr>

3) Logical Expressions

<logic_exp> ::= <expr> and <expr>

| <expr> or <expr>

| not <expr>

| <logic_exp> and <logic_exp>

| <logic_exp> or <logic_exp>

| not <logic_exp>

| <true>

| <false>

| 0

| 1

4) Program Progression

<program> ::= <stmt-list>

<stmt-list> ::= <stmt>

| <stmt-list> <stmt>

<stmt> ::= <assignment_stmt>

| <if_stmt>

| <while_stmt>

| <for_stmt>

| <func_call>

| <decl_stmt>

| <func_def_stmt>

| <input_stmt>

| <output_stmt>

5) If - Else Condition

<if_stmt> ::= <matched_if> | <unmatched_if>

<matched_if> ::= if <expr> then <matched_if> else <matched_if>

| <stmt>

<unmatched_if> ::= if <expr> then <stmt>

| if <expr> then <matched_if> else <unmatched_if>

6) For Statement

<for_stmt> ::= for (<expr>; <expr>; <expr>) <stmt-list>;

7) While Condition

<while_stmt> ::= while (<expr>) <stmt-list>;

8) Functions

<func_call> ::= <func_ident>

<type-ident> ::= int

| float

| char

| void

<ident-list> ::= <ident>

| <ident> , <ident-list>

<decl-stmt> ::= <type-ident> <ident-list> ;

<func_def_stmt> ::= <type-ident> <func_ident> (<ident-list>) <stmt-list>

<primitive_func> ::= <ident>.readInclination()

| <ident>.readAlt()

| <ident>.readTemp()

| <ident>.readAccel()

| <ident>.toggleCamera()

| <ident>.takePic()

| <ident>.readTs()

| <ident>.connect()

9) Input / Output

<input_stmt> ::= input (<ident>)

<output_stmt> ::= output (<string>)

10) Operators

<expr> ::= <expr> <operator> <expr>

| <term>

<operator> ::= + | - | * | /

<term> ::= <term> * <term>

| <term> / <factor>

| <factor>

<factor> ::= <idc> ** <factor> //** is exponentiation

| <idc>

<assignment_stmt> ::= <ident> = <expr> ;

Explanation of … Language Constructs

1) Program Definition

The program definition defines the grammar of the Akame language. Grammar G is consisted of four parts which are V standing for non-terminals and including S start variable, T standing for terminals, P standing for productions.

2) Types and Constants

… language has the following types: Character, digit, sign, boolean expressions and id for constant. Characters are in <char>, <digit> contains digits, <sign> contains + and -, boolean expressions are <true> and <false>.

3) Logical Expressions

Logical expressions contain some of the operators such as and, or, not, true, false, 0 and 1. And, or, not expressions are valid for both expressions and logic expressions.

4) Program Progressions

In this explanation, the working progress of Akame language is analysed. The program is executed with **<program> ::= <stmt-list>** that contains statement lists which have each of the statements with **<stmt-list> ::= <stmt> | <stmt-list> <stmt>.** Then, each of these statements contains assignments such as if, while, for, function calls, declarations, function definitions, inputs and outputs with the following BNF description:

- **<stmt> ::= <assignment_stmt>**
- **| <if_stmt>**
- **| <while_stmt>**
- **| <for_stmt>**
- **| <func_call>**
- **| <decl_stmt>**
- **| <func_def_stmt>**
- **| <input_stmt>**
- **| <output_stmt>**

5) If – Else Condition

- **<if_stmt> ::= <matched_if> | <unmatched_if>**
- **<matched_if> ::= if <expr> then <matched_if> else <matched_if>**
- **| <stmt>**

- **<unmatched_if> ::= if <expr> then <stmt>**
- **| if <expr> then <matched_if> else <unmatched_if>**

If – else conditions are used to determine which operation will be executed. If the condition which are declared as **<matched_if> | <unmatched_if>.** The BNF description of if – else condition is taken from course slides.

6) For Statement

For statement is used in order to execute some statement lists **<stmt-list>** for each time that meets the expressions **<expr>; <expr>; <expr>.** In for, starting expression, ending expression and the operation expression determine the operations on statements list sequentially:

- **for (<expr>; <expr>; <expr>) <stmt-list>;**

7) While Condition

While condition is used in order to execute some statement lists **<stmt-list>** throughout the expression in the while loop has been met. The expression in the while loop is described as follows:

- **while (<expr>) <stmt-list>;**

8) Functions

Functions are whole of specific statements that have specific functions. At first, the function is declared and then it can be called many times within its context.

- **<func_call> ::= <func_ident>**
- **<type-ident> ::= int**

- **| float**
- **| char**
- **| void**
- **<ident-list> ::= <ident>**
- **| <ident> , <ident-list>**
- **<decl-stmt> ::= <type-ident> <ident-list> ;**
- **<func_def_stmt> ::= <type-ident> <func_ident> (<ident-list>) <stmt-list>**
- **<primitive_func> ::= <ident>.readInclination()**
- **| <ident>.readAlt()**
- **| <ident>.readTemp()**
- **| <ident>.readAccel()**
- **| <ident>.toggleCamera()**
- **| <ident>.takePic()**
- **| <ident>.readTs()**
- **| <ident>.connect()**

Function declaration can be seen with **<func_ident>** . In order to determine the return type of the function; int, float, char and void are selected as types. Then, identification lists are composed of identifications and the statement declaration with its type and identification is shown with following statement:

- **<decl-stmt> ::= <type-ident> <ident-list> ;**

Then, function definition statement is created with its type, function identification, identification list standing for function signature and statement list that contains several statements as shown:

- **<func_def_stmt> ::= <type-ident> <func_ident> (<ident-list>) <stmt-list>**

Primitive functions are also declared to read, take a picture, connect etc. because of the drone has these primitive instructions.

9) Input / Output

Inputs take the input as an identification and outputs take the output as a string in the following form:

- **<input_stmt> ::= input (<ident>)**
- **<output_stmt> ::= output (<string>)**

10) Operators

Operators are used in order to execute some operations from logical operations to assignments and defined as following:

- **<expr> ::= <expr> <operator> <expr> | <term>**
- **<operator> ::= + | - | * | /**

<operator> is assigned into the +, -, *, / logical expressions. Then, terms are declared for addition; factor is declared for multiplication, division and the description like:

- **<term> ::= <term> * <term>**
- **| <term> / <factor>**
- **| <factor>**
- **<factor> ::= <idc> ** <factor> //** is exponentiation**
- **| <idc>**
- **<assignment_stmt> ::= <ident> = <expr> ;**

Finally, assignment operator is used with “=” to assign an expression to an identification.

Descriptions of Defined Non – Trivial Tokens

- ASSIGNMENT: Token for assigning expressions
- NEWLINE: Token for creating a newline
- IS_EQUAL: Token for checking the expression is equal to another or not
- MINUS: Token for minus sign and subtraction
- PLUS: Token for plus sign and addition
- IF: Token for conditional if statement
- ELSE: Token for conditional else statement
- ELSE_IF: Token for conditional if – else statement
- COMMENT: Token for comment identification
- HASHTAG: Token for hashtag detection
- FOR: Token for the for statement
- DO: Token for the do statement
- WHILE: Token for the while statement
- GREATER: Token for checking the expression is greater than another
- LESS: Token for checking the expression is less than another
- GTE: Token for checking the expression is greater than another or equal to it
- LTE: Token for checking the expression is less than another or equal to it
- NOT_EQUAL: Token for checking the expression is not equal to another
- SEMICOLON: Token for the semicolon statement
- DOT: Token for the dot statement
- COMMA: Token for the comma statement
- COLON: Token for the colon statement
- MULT: Token for multiplication operation
- DIV: Token for division operation
- OR: Token for or operation
- AND: Token for and operation

- RETURN: Token for return statement in functions
- PRINT: Token for printing out
- NOT: Token for not statement
- BUILTIN_FUNC (readInclination|readAlt|readTemp|readAccel|toggleCamera|takePic|readTs|connect): : Token for the primitive functions for the drone
- INT_TYPE: Token for integer type
- FLOAT_TYPE: Token for float type
- DOUBLE_TYPE: Token for double type
- VOID: Token for void type
- BOOLEAN_TYPE: Token for boolean type

- FUNCTION: Token for function declaration
- LOWERCASE: Token for lowercase words
- UPPERCASE: Token for uppercase words
- TRUE: Token for boolean true statement
- FALSE: Token for boolean false statement
- BOOLEAN: : Token for boolean data type
- ALPHANUMERIC: Token for alphanumeric data type
- IDENTIFIER: Token for identifier data type
- INPUT: Token for input
- STRING: Token for string data type

Test Program of Akame Language

```
for( int i = 0; i < 15; i++ ) {
    double a = 5.2;
    float b = 5.35;
    bool x = true;
    if( (a == b) && x ) {
        print("a == b");
    } elif {
        print( a - b );
        return :!x;
    }
}

while( true ) {
    a.readInclination;
    a.readTemp;
}

function sum(int a, int b) {
    int c = a + b;
    return c;
}
```

--OUTPUT--

COMMENT

FOR LPAR INT_TYPE IDENTIFIER ASSIGNMENT INTEGER SEMICOLON IDENTIFIER LESS INTEGER SEMICOLON IDENTIFIER PLUS PLUS RPAR LBRACKET

DOUBLE_TYPE IDENTIFIER ASSIGNMENT FLOAT SEMICOLON

FLOAT_TYPE IDENTIFIER ASSIGNMENT FLOAT SEMICOLON

BOOLEAN_TYPE IDENTIFIER ASSIGNMENT IDENTIFIER SEMICOLON

IF LPAR LPAR IDENTIFIER IS_EQUAL IDENTIFIER RPAR AND IDENTIFIER RPAR LBRACKET

PRINT LPAR STRING RPAR SEMICOLON

RBRACKET ELSE_IF LBRACKET

PRINT LPAR IDENTIFIER MINUS IDENTIFIER RPAR SEMICOLON

RETURN NOT IDENTIFIER SEMICOLON

RBRACKET

RBRACKET

WHILE LPAR IDENTIFIER RPAR LBRACKET

IDENTIFIER DOT BUILTIN_FUNCSEMICOLON

IDENTIFIER DOT BUILTIN_FUNCSEMICOLON

RBRACKET

FUNCTION IDENTIFIER LPAR INT_TYPE IDENTIFIER COMMA INT_TYPE IDENTIFIER RPAR LBRACKET

INT_TYPE IDENTIFIER ASSIGNMENT IDENTIFIER PLUS IDENTIFIER SEMICOLON

RETURN IDENTIFIER SEMICOLON