Bilkent University
Department of Computer Engineering
CS 315
Programming Languages Project
Part II Report

Team: 58

Language Name: ACEA

Group Members:

Ahmet Cemal Alıcıoğlu	21801700	Section 2
Beril Canbulan	21602648	Section 1
Aziz Ozan Azizoğlu	21401701	Section 2

Instructor: Halil Altay Güvenir

Teaching Assistants: Irmak Türköz, Alper Şahıstan, Duygu Durmuş

Table of Contents

1. Introduction

This report is part of the part 2 of the CS 315: Programming Languages course project.

In this report, we present the improved programming language which is designed specifically for programming drones, ACEA. This report includes the complete BNF description of the grammar of the language along with the detailed explanations for every part. At the end of the report, there exists an evaluation of the language in the means of its writability, readability and its reliability and the conclusion part..

ACEA has a simple syntax that has taken inspiration from many modern day programming languages like C or Python. It combines many great aspects of the popular programming languages to maximize its effectiveness and thus specialize in drone programming.

2. BNF Description

2.1. Program

```
<program> ::= begin <stmts> end

<stmts> ::= <stmt> | <stmt> <stmts> | <comment> <stmts> | <comment>

<stmt> ::= <open_stmt> | <closed_stmt>

<simple_stmt> ::= <assn_stmt> | <assn_declare_stmt> | <return_stmt> |

            <while_stmt> | <input_stmt> | <output_stmt> | <function_stmt> |

            <empty_stmt> | <block_stmt> | <function_definition>

<comment> ::= /* <string> */

<string> ::= <non_empty_string> | <empty>

<non_empty_string> ::= <char> <string> | <char>

<char> ::= <digit> | <letter> | ! | ‘ | ? | ( | ) | * …

<empty> ::= “”
```

2.2. Types

```
<identifier> ::= <identifier><letter> | <letter> | <identifier><digit>

<type_name> ::= num | string
```

<type_declare> ::= <type_name> | final <type_name>

<const> ::= <num_type> | <string_type>

<num_type> ::= [+-]?[0-9]*(\.)? [0-9]+

<string_type> ::= “<string>”

2.3. Expressions

<expression> ::= <expression> <logical_op> <logic> | <logic>

<logic>::= <logic> <relational_op> <relation> | <relation>

<relation> ::= <relation> <low_op> <term> | <term>

<term> ::= <term> <high_op> <factor> | <factor>

<factor> ::= (<expression>) | <identifier> |

        <const> | <function_call> | <builtin_variable> | !<factor>

<high_op> ::= * | / | %

<low_op> ::= + | -

<relational_op> ::= < | > | == | != | <= | >=

<logical_op> ::= “&&” | “||”

2.4. Functions

<function_definition> ::= <type_declare> <function_identifier>(<function_arguments>)
{<stmts>}

<function_identifier> ::= <identifier>

<function_arguments> ::= <non_empty_function_arguments> | <empty>

<non_empty_function_arguments> ::= <type_declare> <identifier> |

        <type_declare> <identifier>, <non_empty_function_arguments>

<function_call_arguments> ::= <empty> | <non_empty_function_call_arguments>

```
<non_empty_function_call_arguments> ::= <expression> | <expression>,
                                         <non_empty_function_call_arguments>

<user_function_call> ::= <function_identifier>(<function_call_arguments>)

<function_call> ::= <user_function_call> | <builtin_function>
```

2.5. Statements

```
<block_stmt> ::= {<stmts>}

<empty_stmt> ::= ;

<assn_stmt> ::= <assn> ;

<assn> ::= <identifier> = <expression>

<assn_declare> ::= <assn> | <identfier>

<multiple_assn_declare> ::= <assn_declare> | <assn_declare> , <multiple_assn_declare>

<assn_declare_stmt> ::= <type_declare> <multiple_assn_declare> ;


<closed_stmt> ::= <simple_stmt> |

        if (<expression>) <closed_stmt> else <closed_stmt> |

        while (<expression>) <closed_stmt> |

        <for_body> <closed_stmt> |

        do <closed_stmt> while (<expression>) ;

<open_stmt> ::= if (<expression>) <simple_stmt> |

       if (<expression>) <open_stmt> |

       if (<expression>) <closed_stmt> else <open_stmt> |

       while (<expression>) <open_stmt> |

       <for_body> <open_stmt> |

       do <open_stmt> while (<expression>) ;
```

```
<for_body> ::= for (<init_stmt> <for_expression> <looping_stmt>)

<init_stmt> ::= <assn_declare_stmt> | <assn_stmt> | <empty_stmt>

<for_expression> ::= <expression> ; | <empty_stmt>

<looping_stmt> ::= <assn> | <empty>


<input_stmt> ::= __scan(<identifier>);

<output_stmt> ::= __print(<expression>);


<function_stmt> ::= <function_call> ;

<return_stmt> ::= return ; | return <expression> ;
```

2.6. Built-in Functions and Variables

```
<builtin_variable> ::= __incline | __altitude | __temperature |
                        __acceleration | __connection

<builtin_function> ::= __up(<expression>) | forward(<expression>) |
                       __left(<expression>) | __right(<expression>) |
                       __back(<expression>) | __down(<expression>) |
                       __rotate_left(<expression>) | __rotate_right(<expression>) |
                       __back_flip() | __front_flip() | __right_flip() | __left_flip() |
                       __land() | __switch_camera() | __take_picture() |
                       __timer_start() | __timer_stop() | __timer_time() | __connect() |
                       __disconnect()
```

3. Language Constructs

3.1 Program

The program is made of many different types of statements. It begins with the reserved word “begin” and ends with “end”.

The convention for this part is that the first two lines of the program are “begin” followed by a blank line. Same applies for the last two lines. The second to last line is a blank line and the last line is just “end”.

Any statement that is not in a function definition is executed.

3.2 Comments

The comments on this language are used between statements and are single line comments. The user must begin comment with “/*” and end it with “*/”. Note that the comment must not have multiple lines.

The convention for comments is that they either come after a statement, or are in their own lines.

Example:

```
num a = 2; /* comment 1 */
/* comment 2 */
```

3.3 Types

Const means a directly typed value. It can have two different types: num and string.

- num means that the value is a float number. Users have to use num type to use both integers and floats. Some examples to a const num are: 4, 2.1, -3.92, +21424, 0
- string type is similar to strings from other languages. Some examples to a const string are: “Hello World!\n” , “a”, “Drone moved forward for 10 meters.\n”

3.4 Expressions

Expression in this language can be multiple things. It can be a const, a variable, a function call, or any operation performed on these.

The arithmetic operations that are available in this language are: summation, addition, multiplication, division; the logical operations that are available are: equals, not equals, greater, greater or equals, less, less or equals.

3.5 Assignment

The syntax of an identifier of a variable is the same as a Java identifier. It must start with an alphabetical character, then it can be followed by an alphanumeric character. For any kind of identifier, the convention is snake_case.

Assignment statement in this language does the same operation as other languages: assigns the expression in the right hand side to the variable in the left hand side.

Users should declare a variable in this language before assigning a value to it. Also, just like C type languages, multiple assignments and declarations can be done in the same line.

Some examples to assignment and declaration statements are as follows:

```
string s = “Hello World!\n”;
num a = 2, b, c = 3;
b = a + b;
```

3.6 Loops:

There are three types of loops in this language: for loop, while loop, or a do_while loop. They are exactly the same as the loops from the C language.

Note that just like C, there is no boolean type in this language so instead of “true” or “false”, 1 or 0 must be used. Any number that isn’t 0 is considered “true”. Also note that there are no break, continue, or goto statements in this language.

3.7 Conditional Statements:

As conditional statements, the usual if and else syntax is used. There is no switch statement in this language.
The convention for if and else statements is the same as the convention of the if statement in C language.

3.8 Input-Output:

In this language input and output are done with special built-in functions.

For taking input, the __scan(<identifier>) function is used. The user must declare the variable name and its type before scanning.
For outputting, the __print(<expression>) function is used. The user can write any expression inside this function and the function prints it into the standard output stream.

Example:

```
num a;
__scan(a);
__print(a*4);
```

Input :

```
2
```

Output:

```
8
```

3.9 Functions:

Users can define and call functions in this language. Before declaring the function, the return type of the function must be used. There is no “void” type available in this language so the convention for functions without return statements is using num before their names and returning 0.

After a function name, function arguments divided by comma may be given between the parentheses. Note that writing type names of the arguments are required to be given for functions as well.

After that, statements are given inside braces to define what this function does. In this area, user may use a return statement to make the function return a value. Function calls can be used in expressions or as a statement.

Recursion is available in this language.

Example:
```
num mul(num a, num b) {
	return a * b;
}

num add_one(num a) {
	a = a + 1;
	return 0;
}

num x = 5, y = 4;
__print(mul(x, y));
__print(“\n”);
add_one(x);
__print(mul(x, y));
__print(“\n”);
```

Output:
```
20
24
```

3.10 Built-in Variables and Functions:

This language contains some built-in functions and variables to control the drone's actions. All of these functions or variables start with two '_' characters.

Built-in variables are variables that indicate some properties of the drone, like its incline, connection or the temperature of the weather.

Built-in functions control the actions of the drone. Of all these functions, only __timer_time() function returns a value. So, except for this function, all functions must be used as a standalone statement.

The units of these variables and functions are as follows:

- __timer_time() returns in seconds.
- __forward(), __up(), __left(), __right(), __back(), __down() take their arguments in centimeters.
- __incline is in degrees.
- __altitude is in centimeters.
- __temperature is in Celsius.
- __acceleration is in meters per second.
- __connection is 1 or 0 depending on the connection of the drone.
- __rotate_left() and __rotate_right() take their arguments in degrees.

3.11 Reserved Words:

"begin" token is reserved for the start of the program.
"end" token is reserved for the end of the program.
"if" token is reserved for if statements.
"else" token is reserved for else statements.
"for" token is reserved for statements.
"do" token is reserved for do-while statements.
"while" token is reserved for while and do-while statements.
"num" token is reserved for the num type.
"string" token is reserved for the string type.
"return" token is reserved for return statements.
The names of built-in functions and variables along with input output functions are reserved.

4. Evaluation of Language

4.1. Readability

Since most of the structures in this language are similar to C, users who are familiar with C syntax will not have a hard time understanding a program in this language.

The fact that this language only has two types creates some problems in reading a program written in this language. "num" type is used for floats, integers, unsigned integers and booleans. Because of this, the reader might need to try harder to understand what a function does since they cannot be sure of how the return value of the language will be used.

However, this problem can easily be avoided if the writer writes comments on their code.

Since this language does not require a main function, it can be said that this language suffers from the same problem as python. The problem is that if the main statements of the program are not together, it can be harder to read what this program does. However this can be solved by better code practises.

Also, since the names of the builtin functions are meaningful and are separable from other user defined functions with the addition of two '_' characters in the beginning of their names, they make this program more readable.

4.2. Writability

In terms of writability, we tried to keep our language more simple to write by just using two variable types: numerical type and string type.

The programmer can declare multiple variables in the same line. Thanks to this, less words need to be typed.

Built-in variables and functions such as "__up", "__forward", "__left", "__right" allow the user to move the drone more easily and functions as "__take_picture", "__switch_camera", "__timer_start", "__timer_stop" allow the user to use features of drone by using a single function. Since these functions are built-in inside the language, programmers don't need to use any library to control drones.

Moreover, since the syntax of our language is similar to C language, the users who are familiar with writing C programs can easily write in this language.

One disadvantage with this language is that its input output statements are not that easy to write like printf() and scanf() from C. Users will have to use __print() function a lot to print complex texts. But we think that this problem should not be solved in the syntax of the language as this can be solved by a library imported function.

This language does not include operations like power, xor, shift, or any of the bitwise operations. To simulate the effects of these operations, one has to find other ways like library imported functions or user defined functions. This will not only slow the computing process, but also affect the writability of the function negatively.

4.3. Reliability

The only problem with the reliability of this programming language is that it has only two data types. Scanning a wrong input and causing the program to crash is more likely in this language than others. Programmers may also get unexpected behaviour from their programs while using "num" type for lots of different purposes. However, this is not a big problem because only inexperienced programmers will not face this issue.

5. Conclusion

This report provides a complete BNF description of ACEA language, its language constructs and evaluation of it in the means of readability, writability and reliability. ACEA language is designed specifically to program drones with its built-in functions and variables for drone programming purposes.

ACEA is designed mostly in C style with some tweaks taken from more modern programming languages like Python.

ACEA is a simple and easy-to-write programming language. Since ACEA is similar to C language, programmers who are familiar with C language write in the language and read the syntax more easily.

However, the simplicity of ACEA creates a problem with its reliability. Because it has only two data types as numerical type and string type inexperienced programmers may easily make mistakes that lead to unexpected functionality or crashes.

ACEA is designed to be simple but effective for drone programming. It may not be the best language for science computations or general purpose programming but we believe it is a powerful tool to program drones with.

6. References

1. Porter H., “Designing Programming Languages for Reliability” (2011), http://web.cecs.pdx.edu/~harry/musings/RelLang.html, accessed October 18 2020.
2. Shalev O., “What determines the readability of a programming language ?”, https://www.quora.com/What-determines-the-readability-of-a-programming-language , accessed October 17 2020.
3. Heckerdorn R., “A Grammar for the C- Programming Language (Version F20)”, (2020), http://marvin.cs.uidaho.edu/Teaching/CS445/c-Grammar.pdf, accessed October 15 2020.
4. “Dangling Else”, Wikipedia, https://en.wikipedia.org/wiki/Dangling_else, accessed November 1 2020.