CS-315 Project 2 Report

Language Name: Lingua Dronum

Pınar Yücel 21802188 Section 3

Cankat Kadim 21802988 Section 3

Ege Moroğlu 21401240 Section 1

BNF Description of Lingua Dronum Language

MAIN "main"

RETURN "return"

DEF "DEF"

CALL "CALL"

WHILE "while"

FOR "for"

DO "do"

IF "if"

ELSE "else"

CONST “const”

READ_INCLINATION "readInclination"

READ_ALTITUDE "readAltitute"

READ_TEMP "readTemperature"

READ_ACCELERATION "readAcceleration"

VIDEO_CAM_ON "videoCameraOn"

TAKE_PIC "takePicture"

READ_TIME_STAMP "readTimeStamp"

CONNECT_TO_BASE_COMPUTER "connectToBaseComputer"

LB "{"

RB "}"

LP "("

RP ")"

ASSIGN_OP "="

ADDITION_OP "+"

SUBTRACTION_OP "-"

MULTIPLICATION_OP "*"

DIVISION_OP "/"

EQUAL_OP "=="

NOT_EQUAL_OP "!="

GREATER_OP ">"

LESS_OP "<"

AND_OP "&&"

OR_OP "||"

NOT_OP "!"

END_STMT ";"

COMMA ","

COMMENT "//"[a-zA-Z0-9 .]*

INPUT_FILE ">>F"

INPUT_KEYBOARD ">>K"

OUTPUT_FILE "<<F"

OUTPUT_CONSOLE "<<C"

DOUBLE ([0-9]*"."[0-9]*|"-"[0-9]*"."[0-9]*)

TEMPERATURE ([0-9]*"."[0-9]*" C"|"-"[0-9]*"."[0-9]*" C")

INTEGER ([1-9][0-9]*|"-"[1-9][0-9]*|"0")

TYPE "int"|"string"|"bool"|"double"|"temp"

BOOLEAN "true"|"false"

FILE_NAME [a-z_][a-zA-Z0-9_]*".txt"

IDENTIFIER [a-z_][a-zA-Z0-9_]*

STRING "\""[a-zA-Z0-9_ :\\n]*"\""

Description of Nontrivial Tokens

COMMENT

The syntax of the comment is simple and foolproof to improve readability and writability. Only letters, digits, spaces and dots are allowed in comments after “//”. The syntax of the comments in this language is similar to the conventional syntax of writing comments in C++ or Java with the difference that only letters, digits, spaces and dots are allowed in comments. There is only one way of writing comments in this language and this feature promotes simplicity and therefore improves readability and writability. Since there is only one possible way of writing comments, programmers using Lingua Dronum can master writing comments easily and therefore would be less likely to make mistakes which enhances the overall reliability of Lingua Dronum.

IDENTIFIER

The syntax of identifiers follows the most common convention in which identifiers start with underscore or a lowercase letter followed by any number of letters, digits or underscores. Identifiers would look complex and cause confusion in some cases if a greater variety of characters were allowed in identifiers. The simple nature of the syntax of identifiers used in this language promotes readability and writability. Enhanced readability and writability of the identifiers together decrease the risk of a programmer using Lingua Dronum to make identifier related mistakes which improves the overall reliability of this language.

Literals (INTEGER, BOOLEAN, STRING, DOUBLE, TEMPERATURE)

The literals in Lingua Dronum are defined by INTEGER, BOOLEAN, STRING, DOUBLE and TEMPERATURE which represent integers, booleans, strings, doubles and temperatures in order. The syntax of the integers and doubles are the same as conventional integers and doubles in C++ or Java. To promote simplicity and therefore enhance readability and writability, only the negativity of the integers and doubles are indicated. If there is no indication of sign, the integer or double is positive. A boolean in this language can either be “true” or “false” which enhances simplicity. Strings can only include letters, digits, underscores, spaces, “\n”s and colons inside quotes. Allowing a limited variety of characters to be used in strings makes strings simpler. TEMPERATURE is a literal type defined for the specific purposes of Lingua Dronum. A temperature is basically a double followed by a space and a “C” which represents celcius. As a result of these simplistic characteristics of literals, they are highly readable and writable and therefore reliable.

Reserved Words (main, return, DEF, CALL, while, for, do, if, else, const, int, string, bool, double, temp, true, false, readInclination, readAltitute, readTemperature, readAcceleration, videoCameraOn, takePicture, readTimeStamp, connectToBaseComputer)

There are only 25 simple reserved words in Lingua Dronum and all of the reserved words have meaningful names which increases the readability and writability of the language. There are no meaningless abbreviations so that a programmer using this language is less likely to make mistakes which increases the reliability of this language.

Input Output

Input and output symbols which are “>>F”, “>>K”, “<<F” and “<<C” are quite simple to write and read and therefore to use. They are similar to the conventional symbols used in C++ for input and output. F stands for file, K stands for keyboard and C stands for console. They are easy to read, write and use. Since they are easy to use, they contribute to the overall reliability of the language.

BNF Description of Lingua Dronum Language

Program Structure

What the program consists of is defined by <program>. There can be function definitions above the <main> which is a convention. In this language, function declerations are not allowed.

<program> ::= <functionDefs><main> | <main>

There can be one or more function definitions defined by <functionDefs>.

<functionDefs> ::= <functionDef> | <functionDef><functionDefs>

Below is the <main> which is defined as conventionally. Each program defined in this language has to have a <main>. Main includes all the statements apart from those that are in function definitions. There can be no statements in the <main> as well.

<main> ::= MAIN LP RP LB<statements>RB | MAIN LP RP LB RB

Statements allowed in this language are listed below.

<statement> ::= <returnStatement> | <variableDec> | <assign>END_STMT | <loop> | <conditionalStatement> |
<input>END_STMT | <output>END_STMT | <functionCall>END_STMT

In this language, <statements> can consist of one or more <statement>.

<statements> ::= <statement>| <statement><statements>

Literals

There are five types of literals

<literal> ::= INTEGER | STRING| BOOLEAN | DOUBLE | TEMPERATURE

Return Statement

The <returnStatement> defines the possible return statements in this language. Only expressions can be returned. Expressions can be simplified to sole identifiers and literals.

<returnStatement> ::= RETURN<expression>END_STMT

Variable Decleration

Variables are declared according to the conventional decleration in this language. To declare a variable, its type and identifier is given followed by end statement symbol. Constant variables must be assigned their values when they are declared.

<variableDec> ::= TYPE IDENTIFIER END_STMT | CONST TYPE IDENTIFIER ASSIGN_OP <expression> END_STMT

Assignment

To assign a variable a value the identifier should be given first followed by the assign operator. After the assign operator, there can be an expression or a function call. There is no END_STMT at the end of <assign> because there are places in this language where <assign> is used without an END_STMT.

<assign> ::= IDENTIFIER ASSIGN_OP<expression>

Expressions

The following table lists the precedence and associativity of Lingua Dronum operators. All of the precedence and associativity rules are according to the conventional C++ rules. can include any operator in any number that is listed in the table below. To ensure the indicated the associativities given, left recursion is used.

Precedence	Operator	Description	Associativity
1	!	logical not	none
2	*	muliplication	left to right
	/	division	
3	+	summation	
	-	subtraction	
4	<	relational operator <	
5	>	relational operator >	
6	==	relational operator ==	
	!=	relational operator !=	
7	&&	logical AND	
8	||	logical OR	

An <expression> can either include an OR_OP or can be an <andExp>.

<expression> ::= <expression>OR_OP<andExp> | <andExp>

An <andExp> can either include an AND_OP or can be an <equalsExp>.

<andExp> ::= <andExp>AND_OP<equalsExp> | <equalsExp>

An <equalsExp> can either include a EQUAL_OP or a NOT_EQUAL_OP or can be an <greaterThanExp>.

<equalsExp> ::= <equalsExp>EQUAL_OP<greaterThanExp> | <equalsExp>NOT_EQUAL_OP<greaterThanExp> |
<greaterThanExp>

A <greaterThanExp> can either include a GREATER_OP or can be an <lessThanExp>.

<greaterThanExp> ::= <greaterThanExp>GREATER_OP<lessThanExp> | <lessThanExp>

A <lessThanExp> can either include a LESS_OP or can be an <additionSubtractionExp>.

<lessThanExp> ::= <lessThanExp>LESS_OP<additionSubtractionExp> | <additionSubtractionExp>

An <additionSubtractionExp> can either include an ADDITION_OP or a SUBTRACTION_OP or can be an <multiplicationDivisionExp>.

<additionSubtractionExp> ::= <additionSubtractionExp>ADDITION_OP<multiplicationDivisionExp>|
<additionSubtractionExp>SUBTRACTION_OP<multiplicationDivisionExp>|
<multiplicationDivisionExp>

A <multiplicationDivisionExp> can either include an MULTIPLICATION_OP or a DIVISION_OP or can be an <parenthesisExp>.

<multiplicationDivisionExp> ::= <multiplicationDivisionExp>MULTIPLICATION_OP<parenthesisExp>|
<multiplicationDivisionExp>DIVISION_OP<parenthesisExp>| <parenthesisExp>

A <parenthesisExp> can either include an <expression> inside parentheses or can be a <notExpression>.

<parenthesisExp> ::= LP<expression>RP | <notExpression>

A <notExpression> can either include a NOT_OP followed by an IDENTIFIER or can be a <literal> or IDENTIFIER. In this language, logical not operator is not allowed before expressions inside parentheses.

<notExpression> ::= NOT_OP IDENTIFIER | <literal> | IDENTIFIER | <functionCall>

Loops

In this language, there are three types of conventional loops. There are more than one loop types to increase the variety of programmes that can be written in this language.

<loop> ::= <whileLoop> | <forLoop> | <doWhileLoop>

The while loop defined in this language starts with the reserved word “while” followed by the expression inside parantheses, followed by statements inside curly braces.

<whileLoop> ::= WHILE LP <expression> RP LB <statements> RB

The for loop defined in this language starts with the reserved word “for” followed by an <assign>, a COMMA, a <expression>, a COMMA and an <assign> inside parantheses, followed by <statements> inside curly braces.

<forLoop>::=FOR LP<assign>COMMA<expression>COMMA<assign>RP LB<statements>RB

The do while loop defined in this language starts with the reserved word “do” followed by statements inside curly braces followed by the reserved word “while” followed by a expression inside parentheses, ending with END_STMT.

<doWhileLoop> ::= DO LB<statements>RB WHILE LP<expression>RP END_STMT

Conditional Statements

A conditional statement in this language can either be an if statement or an if else statement.

<conditionalStatement> ::= <ifStatement> | <ifElseStatement>

<ifStatement> defines the if statement. If statements start with the reserved word “if” followed by a <expression> inside parentheses, followed by statements inside curly braces.

<ifStatement> ::= IF LP<expression>RP LB<statements>RB

<ifElseStatement> defines the if else statement. If else statements start with the reserved word “if” followed by a <expression> inside parentheses, followed by statements inside curly braces, followed by the reserved word “else” followed by <statements> inside curly braces.

<ifElseStatement> ::= IF LP<expression>RP LB<statements>RB ELSE LB<statements>RB

Statements for Input/Output

To read an input into a variable from a file, INPUT_FILE is given first which is “>>F” followed by a FILE_NAME which is the name of the file, followed by an identifier to which the input will be read to. To read an input into a variable from keyboard, INPUT_KEYBOARD is given first which is “>>K” followed by an identifier to which the input will be read to.

<input> ::= INPUT_FILE FILE_NAME IDENTIFIER | INPUT_KEYBOARD IDENTIFIER

To print to a file , OUTPUT_FILE is given first which is “<<F” followed by a FILE_NAME which is the name of the file, and then an identifier, <literal> or a <functionCall> whose value, or whose return value for <functionCall>, will be printed. To print to console , OUTPUT_CONSOLE is given first which is “<<C” followed by an identifier, <literal> or a <functionCall> whose value, or whose return value for <functionCall>, will be printed.

<output> ::= OUTPUT_FILE FILE_NAME IDENTIFIER | OUTPUT_FILE FILE_NAME <literal> |
OUTPUT_FILE FILE_NAME <functionCall> |
OUTPUT_CONSOLE IDENTIFIER | OUTPUT_CONSOLE <literal> | OUTPUT_CONSOLE <functionCall>

Function Definitions and Function Calls

To define a function, reserved word “DEF” is given followed by the type of the return value if there are any, followed by the name of the function which is an IDENTIFIER, followed by an <argsDef> inside parentheses if any, followed by <statements> inside curly braces. Function declarations are not allowed.

<functionDef> ::= DEF TYPE IDENTIFIER LP<argsDef>RP LB<statements>RB |
DEF TYPE IDENTIFIER LP RP LB<statements>RB |
DEF IDENTIFIER LP<argsDef>RP LB<statements>RB |
DEF IDENTIFIER LP RP LB<statements>RB

To call a function the reserved word “CALL” is given followed by the name of the function which is an IDENTIFIER, followed by an <argsCall> inside parentheses if any, followed by <statements> inside curly braces. To call a build-in function, the reserved word “CALL” is given followed by a <buildInFunction>.
<functionCall> ::= CALL IDENTIFIER LP<argsCall>RP | CALL IDENTIFIER LP RP | CALL<buildInFunction>

<argsDef> is the arguments used when defining a function. There can be one or more TYPE followed by IDENTIFIER. If there are more than one TYPE followed by IDENTIFIER, they are seperated with COMMA. This rule is the conventional way of defining the arguments of a function.
<argsDef> ::= TYPE IDENTIFIER | TYPE IDENTIFIER COMMA<argsDef>

<argsCall> is the arguments when calling a function. It consists of one or more identifier. If there are more than one IDENTIFIER, they are seperated by COMMA. This is the conventional way of giving arguments when calling a function.
<argsCall> ::= IDENTIFIER | <literal> | IDENTIFIER COMMA <argsCall> | <literal> COMMA <argsCall>

Primitive Functions for Drone
<buildInFunction> ::= <readInclination> | <readAltitude> | <readTemperature> | <readAcceleration> |
<videoCameraOn> | <takePicture> | <readTimeStamp> | <connectToBaseComputer>

This function returns the current inclination of the drone. It has no arguments. Return type is DOUBLE.
<readInclination> ::= READ_INCLINATION LP RP

This function returns the current altitude of the drone. It has no arguments. Return type is DOUBLE.
<readAltitude> ::= READ_ALTITUDE LP RP

This function returns the current temperature of the drone. It has no arguments. Return type is TEMPERATURE.
<readTemperature> ::= READ_TEMP LP RP

This function returns the current acceleration of the drone. It has no arguments. Return type is DOUBLE.
<readAcceleration> ::= READ_ACCELERATION LP RP

This function turns the drone’s video camera on if the argument is true, turns the camera of the drone off if the argument is false. There is no return statement in this function.
<videoCameraOn> ::= VIDEO_CAM_ON LP BOOLEAN RP | VIDEO_CAM_ON LP IDENTIFIER RP

This function commands the drone to take a picture when called. It has no arguments. There is no return statement in this function.

<takePicture> ::= TAKE_PIC LP RP

This function returns the current time stamp. It has no arguments. Return type is string.

<readTimeStamp> ::= READ_TIME_STAMP LP RP

This function has two STRING arguments that are the name of a wifi to connect and the password of that wifi in order. Returns true if connection was successful, false otherwise.

<connectToBaseComputer> ::= CONNECT_TO_BASE_COMPUTER LP STRING COMMA STRING RP |
CONNECT_TO_BASE_COMPUTER LP IDENTIFIER COMMA IDENTIFIER RP

Conflicts and Ambiguities

No conflict or ambiguity were encountered during any phase of the project.

Evaluation of Lingua Dronum

Lingua Dronum is a language that is analogous to C++ and Java in many ways. Since many programmers are familiar with C++ and Java, Lingua Dronum is a language that programmers would be able to learn quite easily. Overall, there are no meaningless reserved words used in this language which contributes to the readability and writability of this language. Since the reserved words have meaningful names, a programmer using Lingua Dronum is less likely to make a typo mistakes which increases the reliability of the language. Lingua Dronum has three types of loops and two types of conditional statements rather than jump statements. This feature increases the readability and writability of the language since jump statements make programs complex. Again, since it is easy to write loops and conditional statements without jump statements, this feature augments the reliability of the language. Lingua Dronum does not have any feature multiplicity. For each operator defined in the language, there is only one way to accomplish that operator which contributes to the overall simplicity of the language which in turn improves the readability and the writability of the language. Since there is no feature multiplicity, someone new to Lingua Dronum can learn the language more easily in comparison to there being feature multiplicities. When a language is easy to learn, programmers can master the language much more quickly and therefore are less likely to make mistakes which enhances the reliability of the language. Less commonly used symbols such as “£, $, ½, &,” are not allowed to be used in the comments, strings and identifiers to prevent confusion and promote simplicity since simplicity results in improved readability and writability which in turn results in improved reliability