CS 315
Programming Languages

Project 2 Report

Dropcy

Team 34
Fatma Sena Genç - 21901426 - Section 1
Mustafa Efe Tamyapar - 21902856 - Section 1

Table of Contents

1. Complete BNF Description of Dropcy

```
<program>->flight<LP><string><RP><LCB><statements>flight_end<RCB>

<statements>-><statement>
           |<statement><statements>
<statement>-><matched>
           |<unmatched>
<matched>-><loops>
           |<function_declare>

           |<function_call>
           |<input>
           |<output>
           |<variable_declaration>
           |<variable_assign_value>
           |<variable_declare_initalize>
           |<comments>
           |if<LP><logic_exp><RP><LCB><matched_statement><RCB>else
           <LCB><matched><RCB>
<unmatched>->if<LP><logic_exp><RP><LCB><statements><RCB>
           |if<LP><logic_exp><RP><LCB><matched_statement><RCB>else<L
           CB><unmatched><RCB>
<matched_statement>-><matched>|<matched_statement><matched>
<function_declare>-><return_type><identifier><LP>
<function_parameters><RP>enter<statements>return<return_var>exit
                |<return_type><identifier><LP><RP>enter<statements>r
                eturn<return_var>exit

<function_call>->call<function_name><LP>
           <function_arguments><RP>
           |call<function_name><LP><RP>
<return_type>-><variable_types>
           |<void>
<return_var>-><identifier>
           |<void>
<function_parameter>-><variable_types><identifier>
<function_parameters>-><function_parameter>
                |<function_parameter><function_parameters>
<function_argument>-><identifier>
<function_arguments>-><function_argument>
```

```
           |<function_argument><function_arguments>
<function_name>-><primitive_function_names>
           |<identifier>
<primitive_function_names>->getHeading
                           |getAltitude
                           |getTemperature
                           |goUp
                           |goDown
                           |verticalStop
                           |goForward
                           |goBackward
                           |horizontalStop
                           |turnLeft
                           |turnRight
                           |nozzleOn
                           |nozzleOff
                           |wifiConnect
                           |wifiDisconnect
                           |endTimer
                           |startTime
<comments>-><comment_face><string>
<comment_face>-><LP><multiplication_op><comma><multiplication_op>
<RP><divison_op>
<expressions>-><arithmetic_exp>
                |<logic_exp>
                |<function_call>
<logic_exp>-><relational_exp>
                |<boolean_exp>
<arithmetic_exp>-><arithmetic_exp><plus><terms>
                |<arithmetic_exp><plus><int>
                |<arithmetic_exp><minus><terms>
                |<arithmetic_exp><minus><int>
                |<identifier><plus><terms>
                |<identifier><plus><int>
                |<int><plus><identifier>
                |<identifier><minus><terms>
                |<int><minus><identifier>
                |<identifier><minus><int>
                |<identifier><plus><identifier>
                |<identifier><minus><identifier>
                |<terms>
```

```
<terms>-><terms><multiplication_op><component>
           |<terms><multiplication_op><int>
           |<terms><division_op><component>
           |<terms><division_op><int>
           |<term><modulo_op><component>
           |<term><modulo_op><int>
<component>-><LP><arithmetic_exp><RP>
<relational_exp>-><variable_values><relational_op>
           <variable_values>
           |<identifier><relational_op><identifier>
           |<identifier><relational_op><variable_values>
           |<arithmetic_exp><relational_op><arithmeric_exp>
           |<identifier><relational_op><arithmeric_exp>
           |<arithmetic_exp><relational_op><variable_values>
<boolean_exp> -> <relational_exp><logic_op><boolean_exp>
|<exclamation_m><logic_exp><colon>
<digit> -> 0|1|2|3|4|5|6|7|8|9
<char> -> a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
           |A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<char_for_identifier> -> <char>|<underscore>
<string_components> ->
|!|"|#|$|%|&|'|(|)|*|+|,|-|.|/|0|1|2|3|4|5|6|7|8|9
|:|;|<|=|>|?|@|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|[
|\|]|^|_|`|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|{|||}
|~
<string> -> “<string_components>|<string><string_components>”
<int> -> <digit>|<int><digit>
<identifier_components> -> <char_for_identifier>|<int>

<identifier> -> #<identifier_components>
                |#<identifier><identifier_components>
<boolean> -> <true>|<false>
<heading> -> <int>
<altitude> -> <int>
<temperature> -> <int>
<variable_types> -> int|char|string|boolean|heading|altitude
                |temperature|timer
<variable_declaration> -> create<variable_types><identifier>
<variable_declare_initialize> -> create<variable_types>
           <identifier><assignment_op><variable_values>
           |create<variable_types><identifier>
           <assignment_op><expressions>
<variable_assign_value> -> make<identifier><assignment_op>
                     <variable_values>
                     |make<identifier><assignment_op>
                     <expressions>
<variable_values> -> <int>|<char>|<string>
                     |<boolean>
<input> -> input<tilde><identifier>
<output> -> output<tilde><variable_values>|output<tilde><identifier>
<loops> -> <while_loop>|<for_loop>
<while_loop> -> while<LP><logic_exp><RP><LCB><statements><RCB>
<for_loop> -> for<LP><variable_declare_initialize>
                <semicolon><logic_exp><semicolon><statements><RP>
                <LCB><statements><RCB>

<dot> -> .
<underscore> -> _
<semicolon> -> ;
<colon> -> :
<comma> -> ,
<exclamation_m> -> !
<question_m> -> ?
<GT> -> >
<LT> -> <
<LTE> -> <=
<GTE> -> >=
<equal_check> -> ==
<and_op> -> &&
<or_op> -> ||
<multiplication_op> -> *
<division_op> -> /
<plus> -> +
<minus> -> -
<and_sign> -> &
<pipe_sign> -> |
<true> -> true
<false> -> false
<apostrophe> -> ‘
<modulo_op> -> %
<assignment_op> -> =
<exponention_op> -> ^
```

<tilde> -> ~
<LP> -> (
<RP> ->)
<quotation_m> -> “
<LCB> -> {
<RCB> -> }
<relational_op> -> <LT>|<GT>|<GTE>|<LTE>|<equal_check>
<logic_op> -> <and_op>|<or_op>|<exclamation_m>

2. Language Constructs

Variable Identifiers

In the programming language Dropcy we have variable identifiers:int,char,string,boolean,heading,timer,altitude and temperature. While creating the grammar of the language we think of all of the possible cases where a user needs to use such a kind of a variable identifier. Heading, altitude,timer and temperature are the ones especially for the Drone's physical compatibility. Heading, altitude and temperature uses integers. Strings are used with quotation marks and combinations of all the characters in ASCII. Moreover, Dropcy uses **create** reserved words for variable creation.
Also in dropcy to differentiate identifiers # (hashtag) before identifiers.

Assignment Operator

For assignment operator Dropcy uses **make** reserved word and assignment operator after the variable identifier. Assignment operator "=", uses the value of the variable which is in the right hand side and assigns that value to the identifier which is in the left hand side.

Precedence and associativity of the operators

In Dropcy for precedence, in arithmetic expressions, division, multiplication and modulus operations have precedence over addition and subtraction operations. For the associativity, subtraction and division operators are left associative. For this we made division and subtraction operations grammar left recursive.

Expressions

In the programming language Dropcy, there are four kinds of expressions. Arithmetic expression, logic expression, relational expression and boolean expressions. In arithmetic expressions, arithmetic operations + , - , * , / , % are used and division operation , multiplication and modulus operations has precedence on addition, subtraction operations.In logic expressions, we used & , | , ! operations . We use them to combine expressions. In relational expressions , relational operations < ,> , >= , <= , = are used. In this expressions we compare two sides of the operator. Lastly in boolean expressions we use true and false to indicate boolean expressions.Also in our grammar when user wants to indicate a expression is not they add : (colon) at the and of the expression. In the grammar of the expressions we used boolean and relational expressions inside the logic expression to be able to use them together.

Loops

In Dropcy there are two kinds of loops while and for. In while loop inside the parentheses the user gives a logic expression inside of the parentheses after while word. After parentheses user gives the statements which they want to iterate inside the curly brackets. In the for loop, the user initializes a variable inside the parentheses after the for word. Then the user uses a semicolon , gives a logical expression, uses a semicolon and gives a statement, uses a semicolon , closes the parenthesis . After the parentheses the user uses curly braces and gives the statements which they want to iterate inside the braces.

Conditional Statements

For conditional statements if and else statements are used. In grammar we check if a statement is matched or not to increase the reliability of the program. User uses if, else reserved words after this gives the logic expression inside the parentheses. After the line user gives the statements which they want inside if, else.

Input/Output Statements

In Dropcy for input we use **input** reserved word and a tilde(~) after to create an input. Input is used for any value wanted from the user For the output we use the **output** reserved word and a tilde(~) after to create an output. Output is used for printing the wanted statement.

Function Definitions and Function Calls

For function definitions the user first initiates the return type after that gives the function name and parameter of the function inside parentheses. Functions can have no parameter. After the parameter,**enter** reserved word is used for the start of the function . User gives the statements which are needed for the function and then finishes the function with **exit** reserved word. If the function is not void, the user gives the return type using **return** reserved word. For the function calls, the user uses **call** reserved word that gives the name of the function and parameters of the function inside the parentheses.

Comments

For comments, we used **(*,*)/** face notation to start the comment. Face notation is followed by a string variable. String consists of all of the ASCII characters and their combinations.

Primitive Functions

In the Dropcy programming language there are 15 primitive functions. **getHeading**, **getAltitude**,**getTemperature** are getter functions of the identifiers. **goUp**,**goDown**,**verticalStop**, **goForward**, **goBackward**, **horizontalStop**, **turnLeft**, **turnRight** are functions for Drone movement. Also **endTimer,startTime** functions are added to measure the time in Drone. These functions are used in especially calculating the travel distance. Lastly, **nozzleOn**, **nozzleOff**, **wifiConnect**,**wifiDisconnect** are used for controlling nozzle and wifi connection of the drone. We tried to come up with the most number of primitive functions for the drone.

3. Reserved Words

flight : Word that starts programs
flight_end : Word that finishes programs
if : Word used in if statements
else : Word used in else statements
enter : Word that is used to start function declaration
exit : Word that is used to finish a function declaration
return :Word that is used before return type
call : Word that is used for function calls
create : Word that is used while creating value
make : Word that is used while assigning value to a variable
input : Word that is used before getting an input
output: Word that is used before output
while : Word that is used to indicate while loop
for : Word that is used to indicate for loop
true : Word that is used to indicate true boolean type
false : Word that is used to indicate false boolean
Reserved Words for our variable types:
int
char
string
boolean
heading
altitude
temperature
timer

4. Nontrivials

<program> : this is the start token of the language
<statements> : statements involves multiple statements that can be written inside the program
<statement> : singular statements that involve matched or unmatched statements
<matched> : all possible statements except if-else ones (but it includes if-else involving matched statements)
<unmatched> : if-else statements with if involving matched or unmatched statements & else involving unmatched statements
<matched_statement> : singular or multiple matched statements
<function_declare> : various possible statements to create functions

<function_call> : statements to call functions
<return_type> : possible function return types
<return_var> :this token involves what can be written next to "return" reserved word
<function_parameter> : singular function parameter construct
<**function_parameters>** : multiple function parameters
<function_argument> : any identifier can be an argument (since identifiers can be variable names)
<function_arguments> : multiple function arguments
<function_name> : primitive function names, or a function name that user specifies
<primitive_function_names> : names of all the primitive functions in the language
<comments> : comment statement
<comment_face> : the expression that is required before a comment statement. Without this expression, the comment cannot be identified
<expressions> : possible expressions (arithmetic, logic, or function calls
<logic_exp> : relational or boolean expressions
<arithmetic_exp> : all possible arithmetic expression combinations, involving adding or subtracting. Also, can be "terms" to provide operator precedence
<terms>-> : all possible multiplication and division expression combinations
<component>-> : arithmetic expression in parenthesis (used to provide parenthesis precedence)
<relational_exp> : all possible relational expressions

<boolean_exp> : all possible boolean expressions
<digit> : all possible digits
<char> : all alphabetic chars, lowercase and uppercase
<char_for_identifier> : character components for an identifier, alphabetic chars and underscore
<string_components> : all possible character
<string> : singular or multiple string components, they must be written between quotation marks
<int> : integer values
<identifier_components> : components that can make up an identifier - character components and integers

<identifier> : a valid identifier name, preceded by a hashtag symbol
<boolean> : values boolean can take - true or false
<heading> : values heading can take - int
<altitude> : values altitude can take - int
<temperature> : values temperature can take - int
<variable_types> : possible variable types. They are used when variables are being created
<variable_declaration> : expression to create variable
<variable_declare_initialize> : expressions to create a variable and give it a value
<variable_assign_value> : expressions to assign value to already created variables
<variable_values> : possible variable value types
<input> : expression to get input
<output> : expressions to print output
<loops> : possible loop types - for and while
<while_loop> : while loop statement
<for_loop> : for loop statements
<relational_op> : possible relational operators
<logic_op> : possible logic operators