HACETTEPE UNIVERSITY

Department Of Computer Engineering

BBM301 Programming Languages

Project Report

Team Members :

- **21727743 Muhammet Eren Taşdemir**
- **21627213 Oğuzhan Eroğlu**
- **21627758 İhsan Kürşad Ünal**

Tokens :

1. **START :** Indicates that the program has started
2. **FINISH:** Indicates that the program has ended
3. **INPUT :** Represents taken parameters from input units like keyboard , mouse , etc. . Matches with ‘input’ keyword.
4. **PRINT :** This token represents actual print (writing to screen) mission . Matches with ‘print’ keyword
5. **PRINTSTRING:** Represents a list of any char between quotes.
6. **GRAPH :** A data type , matches when ‘graph’ typed with capital G.
7. **DEF :** While declaring a new function this keyword ‘def’ indicates that declaration
8. **IF :** Matches with ‘if’ keyword . Used to represent if statement.
9. **ELSE :** Matches with ‘else’ keyword. If if statement doesn’t satisfy then this statement acts like a unconditioned if statement. Of course only condition “previous if statement doesn’t satisfy”.
10. **WHILE :** Matches with ‘while’ keyword. Used for ‘while statement’.
11. **FOR** : Matches with ‘for’ keyword . Used for ‘for statement’.
12. **INT_DECLARE :** Matches with ‘int’ keyword . Used for integer typed declarations.
13. **FLOAT_DECLARE :** Matches with ‘float’ keyword . Used for floating point typed declarations.
14. **CHAR_DECLARE:** Matches with ‘char’ keyword . Used for character typed declarations.
15. **SHOWONMAP :** Matches with ‘SHOWONMAP’ keyword . It is a build-in function
16. **SEARCHLOCATION:** Matches with ‘SEARCHLOCATION’ keyword . It is a build-in function
17. **GETROADSPEED :**Matches with ‘GETROADSPEED’ keyword . It is a build-in function
18. **GETLOCATİON :** Matches with ‘GETLOCATION’ keyword . It is a build-in function
19. **SHOWTARGET :** Matches with ‘SHOWTARGET’ keyword . It is a build-in function
20. **ADDROAD :** Matches with ‘ADDROAD’ keyword . It is a build-in function
21. **SHOWCROSSROADS :** Matches with ‘SHOWCROSSROADS’ keyword . It is a build-in function.
22. **SHOWROADS :** Matches with ‘SHOWROADS’ keyword . It is a build-in function

23. **COMMA :** Matches with ‘,’ .
24. **SEMICOLON :** Matches with ‘;’ .

25. **COLON :** Matches with ‘:’ .
26. **INT :** Matches with any typed integer value like 1,2,4 and matches with signed integer values like +5 or -7.
27. **FLOAT :** Matches with any typed floating point value like 1.1 , 2.0 and matches with signed floating point values like +5.4 or like -1.4 also matches values like .9 , .67.
28. **COMMENT :** Represents comment line . Starts with sharp symbol and ends with in again. Between these two sharp symbols locate any character or character list.
29. **VAR_NAME :** Variables take any name with these restrictions :
    - Variable can start only with lowercase letters
    - Then it can continue with lower or upper case letters or digits , starts from zero to nine , or underscore character
30. **FUNC_NAME :** Function can takes and name with same restrictions as VAR_NAME except FUNC_NAME have to start with uppercase letter.
31. **ARRAY :** It is actually int array or floating point array. Meas that list of integers (or list of floating point numbers) between square brackets.
32. **PLUS :** Arithmetic addition operator . Matches with ‘+’ character
33. **MINUS :** Arithmetic subtraction operator . Matches with ‘-’ character
34. **MULTIPLY :** Arithmetic multiplication operator . Matches with ‘*’ character
35. **DIVIDE :** Arithmetic division operator . Matches with ‘/’ character
36. **OPEN_PAR :** Matches with special character for our language ‘(‘.
37. **CLOSE_PAR :** Matches with special character for our language ‘)‘.
38. **OPEN_SQR_BRC :** Matches with special character for our language ‘[‘.
39. **CLOSE_SQL_BRC :** Matches with special character for our language ‘]‘.
40. **OPEN_CURLY_PAR :** Matches with special character for our language ‘{‘.
41. **CLOSE_CURLY_PAR :** Matches with special character for our language ‘}‘.
42. **OR :** Matches with ‘||’ character tuple as a logical operator.
43. **AND :** Matches with ‘&&’ character tuple as a logical operator
44. **ASSIGN :** Matches with ‘=’ character . Represents left hand side equals right hand side
45. **EQUAL :** Matches with ‘==’ character tuple as a logical operator.
46. **NOTEQUAL :** Matches with ‘!=’ character tuple as a logical operator.
47. **LESSOREQUAL :** Matches with ‘<=’ character tuple as a relational comparison operator
48. **GREATEROREQUAL :** Matches with ‘>=’ character tuple as a relational comparison operator
49. **LESSTHAN :** Matches with ‘<’ character as a relational comparison operator
50. **GREATERTHAN :** Matches with ‘>’ character as a relational comparison operator
51. **INVALID :** Characters or character sets without out declarations are invalid

Also we check number of lines of given file.

Grammar

<program> : START <statements> FINISH →Program is a body with name and without declaration ; starts with ‘START’ keyword and finishes with ‘FINISH’ keyword

<statements> : <statement> <statements> | <statement> → Every expression , block , etc all of them individually is a statement . Statement non-terminal has right associativity. This associativity saves expression from ambiguity.

<statement> : <non_block_st> | <block_st> | COMMENT → Every thing is a subset of statement . DiVides to three main parts .

<non_block_st> : <assign> SEMICOLON | <var_def> SEMICOLON | <return> SEMICOLON |

<func_call> SEMICOLON | <print> SEMICOLON | <graph> SEMICOLON|

<scan> SEMICOLON | <array_def> SEMICOLON → Non-block means ; statements that can express on a one line. Do not need additional statements to express. Ends with semicolon ‘;’.

<block_st> : <if_stmt>| <while> | <for> | <func_dec> → Block is a field that express a thing with more than one statement or zero.

<var_def> : <type> VAR_NAME → We need some named values . And this name should say us what it holds roughly like what that values type. Actually we will not see this type but name will make a link between type and actual value.

<graph> : GRAPH FUNC_NAME OPEN_PAR <parameters> CLOSE_PAR→A data type declaration. Will hold whole map.

<array_def> : <type> VAR_NAME ARRAY → We have two type array int and float specify which one give a name but ARRAY token will make it a array. Actually open and close square brackets

<type> : INT_DECLARE | FLOAT_DECLARE | CHAR_DECLARE → Used to explicit declaration

<return> : RETURN <rhs> →This display type makes return statement more general. It can return any thing except declarations and blocked statements.

<print> : PRINT <rhs> →This display type makes print statement more general. It can print any thing except declarations and blocked statements.

<scan> : INPUT <rhs> →This display type makes scan statement more general. It can take as input an any thing except declarations and blocked statements.

<func_dec> : <type> DEF FUNC_NAME OPEN_PAR <parameters > CLOSE_PAR <body> | <built_in_funcs> → While declaring a function fist specify return type then give a name (FUNC_NAME) after type parentheses and between these parentheses give and specify input parameters and types now you can open body.

<built_in_funcs> : <show_on_map> | <search_location> | <get_road_speed> | <get_location> | <show_target> | <add_road> | <show_crossroads> | <show_roads>→Collect on build-in functions

<show_on_map> : SHOWONMAP OPEN_PAR <var> COMMA <var> CLOSE_PAR SEMICOLON →These build-in functions doesn't need declaration , just call. This is shows how to call build-in functions . This build-in function used to show user location on map

<search_location> : SEARCHLOCATION OPEN_PAR ARRAY CLOSE_PAR SEMICOLON | SEARCHLOCATION OPEN_PAR VAR_NAME CLOSE_PAR SEMICOLON→These build-in functions doesn't need declaration , just call. This is shows how to call build-in functions . This build-in function used to search specific location on map

<get_road_speed> : GETROADSPEED OPEN_PAR GRAPH CLOSE_PAR SEMICOLON | GETROADSPEED OPEN_PAR VAR_NAME CLOSE_PAR SEMICOLON→These build-in functions doesn't need declaration , just call. This is shows how to call build-in functions. This build-in function used to get what is the speed limit on specific road on map

<get_location> : GETLOCATION OPEN_PAR ARRAY CLOSE_PAR SEMICOLON | GETLOCATION OPEN_PAR VAR_NAME CLOSE_PAR SEMICOLON→These build-in functions doesn't need declaration , just call. This is shows how to call build-in functions . This build-in function used to get location of a map element or user

<show_target> : SHOWTARGET OPEN_PAR ARRAY CLOSE_PAR SEMICOLON | SHOWTARGET OPEN_PAR VAR_NAME CLOSE_PAR SEMICOLON→These build-in functions doesn't need declaration , just call. This is shows how to call build-in functions . This build-in function used to show specific target on map. Of course first searches then gets location after shows it.

<add_road> : ADDROAD OPEN_PAR GRAPH COMMA ARRAY CLOSE_PAR SEMICOLON | ADDROAD OPEN_PAR VAR_NAME COMMA VAR_NAME CLOSE_PAR SEMICOLON→These build-in functions doesn't need declaration , just call. This is shows how to call build-in functions . This build-in function used to add new road to map . Actually adds new edge to the Graph

<show_crossroads> : SHOWCROSSROADS OPEN_PAR GRAPH CLOSE_PAR SEMICOLON | SHOWCROSSROADS OPEN_PAR VAR_NAME CLOSE_PAR SEMICOLON→These build-in functions doesn't need declaration , just call. This is shows how to call build-in functions . This build-in function used to show crossroads meas vertexes on Graph.

<show_roads> : SHOWROADS OPEN_PAR GRAPH CLOSE_PAR SEMICOLON | SHOWROADS OPEN_PAR VAR_NAME CLOSE_PAR SEMICOLON→These build-in functions doesn't need declaration , just call. This is shows how to call build-in functions . This build-in function used to show roads on map means edges on Graph

<func_call> : FUNC_NAME OPEN_PAR <parameters> CLOSE_PAR →While calling declarated function give function name and between parentheses give input parameters.

<parameters> : <parameter> COMMA <parameters> | <parameter> →Parameters separates each other with commas and this separation is right associative. This saves us from ambiguity

<parameter> : <var> →Parameters are variable.

<if_stmt> : <matched> | <unmatched> →There is two type if statement matched and unmatched : matched meas if-else tuple and unmatched means just if statement

<matched> : IF <cond_expr> COLON <matched> ELSE COLON <matched>|

<body> →Matched is also body we can put body instead of matched for single if-else match but if it is more complex we need additional non-terminal unmatched

<unmatched> : IF <cond_expr> COLON <if_stmt> |

IF <cond_expr> COLON <matched> ELSE COLON <unmatched> → Just if or nested ifs this non-terminal catch up to help.

<while> : WHILE <cond_expr> <body> →Standard check-do expression. Checks relatively to conditional expression then ‘do’ if condition is true . Do part in body.

<for> : FOR OPEN_PAR <assign> SEMICOLON <cond_expr> SEMICOLON <assign> CLOSE_PAR <body> →Standard for expression . Fist initializes increment(or decrement) value while conditional expression is true do in the body (before or after) then update increment(or decrement) value.

<body> : OPEN_CURLY_PAR <statements> CLOSE_CURLY_PAR →With curly brackets show it is a body . In this body may not be any statement.

<assign> : <lhs> ASSIGN <rhs> →Make left hand side equal with right hand side. Than what is assign mean . Left hand side may var_def (or array _def both are declaration) or may be declarated variable.

<lhs> : <var> | <var_def> | <array_def> → Left hand side may var_def (or array _def both are declarations) or may be declarated variable.

<rhs> : <cond_expr> | OPEN_CURLY_PAR <parameters> CLOSE_CURLY_PAR → Right hand side may logic expression may be arithmetic expression or function call . For initialize multi valued data type like array or Graph give input parameters between curly brackets.

<cond_expr> : <or> →Conditional expression starts with or logic operation because or less prior operation.

<or >: <and> | <or> OR <and> →Variable based or operator

<and> : <eq> | <and> AND <eq> →Variable based and operator

<eq> : <cmp> | <eq> EQUAL <cmp> | <eq> NOTEQUAL <cmp> →Variable based equality operators

<cmp> : <expr> | <cmp> LESSTHAN <expr> | <cmp> GREATERTHAN <expr> | <cmp> LESSOREQUAL <expr> | <cmp> GREATEROREQUAL <expr>
→Variable based relational comparison operators

<expr> : <expr> PLUS <expr2>

| <expr> MINUS <expr2>

| <expr2> →Variable based addition/subtraction operator

<expr2> : <expr2> MULTIPLY <expr3>

| <expr2> DIVIDE <expr3>

| <expr3> →Variable based multiplication/division operator

<expr3> : OPEN_PAR <expr> CLOSE_PAR

| <var> →Variable between parentheses highest precedence

<var> : VAR_NAME | INT | FLOAT | <func_call> | PRINTSTRING → Variables can are rhs expressions means they can be and thing except declarations or bodies . Any thing can ret

Tutorial

- In this language, all codes start with the “START” command and end with the “FINISH” command.

Exp:

START

\# some code \#

FINISH

- Each line of code ends with “;” character.
- Comment lines are written between “\#” characters.
- Variable types are integer or float or char.
- Variables can be declared as type varname; E

Exp: int x; or float y; etc.

- Variables can be initialized when defining like type varname = value;

Exp: int x = 3; etc.

- Arrays are declared like that: type array[] or type array[][] or type array[][][]. Respectively; 1 dimensional, 2 dimensional and 3 dimensional.

Exp: int array[2][3][2];

- print function takes parameters like variable, integer, float, char, string or function call.

Exp: print a; print 3; print “Hello World”; print Swap(5,9);

- return function takes parameters like variable, integer, float, char, string or function call or list of them.

Exp: return {a}; return {3}; return {“Hello World”}; return {Swap(5,9)};

- input function takes variable parameter.

Exp: input x;

- Assigns the value entered by the user to the variable.
- Assignment operation is defined as lhs = rhs;

Exp: x = y; x=3; x= func(3); x = True ;

- Lhs can be variable or variable definition or array definition.
- Rhs can be conditional expression or variable or string, or array or char or float or int.
- In this language we have arithmetic or conditional operations like +, -, /, *, ==, !=, >, <, <=, >=, ||, &&
- Defining functions: type def func_name(parameter or parameter_list){
- block
- }

Exp:

```
int def Swap(x,y){
    a = x;
    b = y;
    x = b;
    y = a;
    return{x,y};
}
```

- Defining while loop:

```
while cond_exp {
block
}
```

Exp:

```
float counter = 5.5;
while a > 10 {
    print a;
    a = a + 1;
}
```

- Defining for loop:

For(initialize variable; cond_exp; assignment){

block

}

Exp:

```
int counter2;
for(counter2 = 0; counter2 < 10 ; counter2 = counter2 + 2){
    input x;
}
```

- İf can be defined as;

İf cond_expr:{

Block(can be other if-else or if statement)

}

*optional*Else:{

Block(can be other if-else or if statement)

}