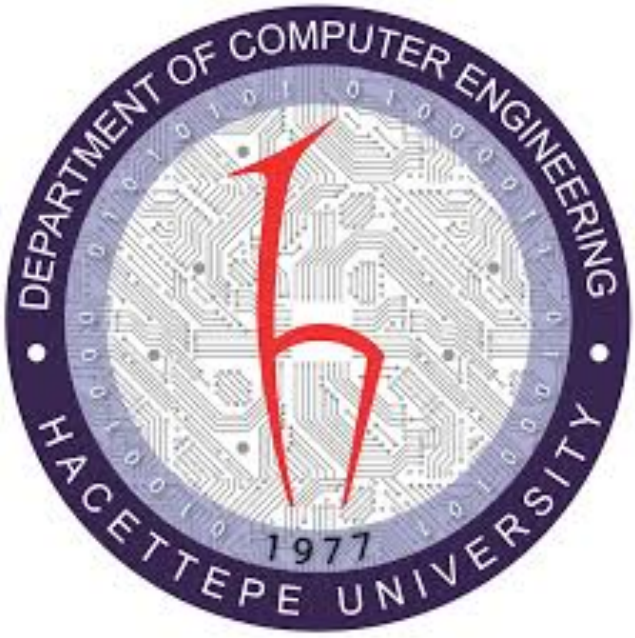
HACETTEPE UNIVERSITY

BBM 301 : PROGRAMMING LANGUAGES



Course Project

Team Members

Deniz Ece Aktaş	21626901
İrem Dereli	21727135
Ece Omurtay	21627543

SHORT DESCRIPTIONS OF THE DEFINED TOKENS

Tokens:

- **BLTIN_SHOW_ON_MAP** : ShowOnMap

This token takes two parameters which are longitude and latitude. It shows the specific location on map with respect to longitude and latitude values.

- **BLTIN_SEARCH_LOCATION** : SearchLocation

This token takes address as argument. It searches the address on map. If search returns true, it marks the address on map.

- **BLTIN_GET_ROAD_SPEED** : GetRoadSpeed

This token finds the road speed.

- **BLTIN_GET_LOCATION** : GetLocation

This token finds the user's location on map and marks.

- **BLTIN_SHOW_TARGET** : ShowTarget

This token shows the specified address' location on map.

- **BLTIN_SHOW_CROSS_ROAD** : ShowCrossRoad

This token finds the location of road on map and determines if specified road is crossroad or not. It returns true if road is crossroad and false if road is not crossroad. It takes road as parameter.

- **BLTIN _PRINT** : Print

This token prints arguments to screen.

- **BLTIN_FIND_DESTINATION** : FindDestination

This token takes two arguments which are longitude and latitude. It finds the destination location on map.

OPERATOR_EQUAL : = This token is equal operator

OPERATOR_SUM : + This token is sum operator

OPERATOR_SUB : - This token is subtract operator

OPERATOR_DIV : / This token is division operator

COMPARATOR_NOT_EQUAL : != This operator checks if two values are not equal

COMPARATOR_EQUAL : == This token checks if two values are equal

COMPARATOR_GREATER : > This token checks if a value is greater than the other

COMPARATOR_SMALLER : < This token checks if a value is smaller than the other

COMPARATOR_GREATER_EQUAL : >= This token checks if a value is greater than or equal to the other value

COMPARATOR_SMALLER_EQUAL : <= This token checks if a value is smaller than or equal to the other value

LEFT_PAR : (This token is the left paranthese

RIGHT_PAR :) This token is the right paranthese

COMMA : , This token is the comma

DOT : . This token is the dot

COLON : : This token is the colon

SEMICOLON : ; This token is the semi colon

TOKEN_IS : IS This token is used in assign statements

TOKEN_RETURN : RETURN This token is used in return statements

TOKEN_NOTHING : NOTHING This token is used in return statements

TOKEN_IF : IF This token is used in if statements

TOKEN_ELSE : ELSE This token is used in if statements

TOKEN_ENDIF : ENDIF This token is used to end the if statements

TOKEN_WHILE : WHILE This token is used in while statements

TOKEN_END_WHILE : ENDWHILE This token is used to end the while statements

TOKEN_FOR : FOR This token is used in for statements

TOKEN_END_FOR	:	ENDFOR	This token is used for end the for statements
TOKEN_START	:	START	This token is used to start the program
TOKEN_END	:	END	This token is used to end the program
BOOLEAN	:	TRUE or FALSE	This token is the boolean expression and it can be true or false
AND	:	AND	This token is the and statement
OR	:	OR	This token is the or statement
NOT	:	NOT	This token is the not statement
INTEGER	:	INTEGER	This token is numbers from zero to infinity
VARIABLE	:	VARIABLE	This token is a single lower case letter or a lower case string
IDENTIFIER	:	IDENTIFIER	This token starts with an uppercase letter and can be followed by a single or many lower or upper letters
STRING	:	STRING	This token is all the numbers and letters
INT_TYPE	:	INT	This token is type of int
STRING_TYPE	:	STR	This token is type of string
ARRAY_TYPE	:	ARRAY	This token is type of array
GRAPH_TYPE	:	GRAPH	This token is type of graph
IRCE_ADDRESS	:	ADDRESS	This token is type of address
IRCE_ROAD	:	ROAD	This token is type of road
IRCE_USER	:	USER	This token is type of user
IRCE_CROSSROAD	:	CROSSROAD	This token is type of crossroad
IRCE_LONGITUDE	:	LONGITUDE	This token is type of longitude
IRCE_LATITUDE	:	LATITUDE	This token is type of latitude
IRCE_FUNC	:	FUNC	This token is used for function statement

IRCE_END_FUNC : ENDFUNC This token is used for end the function statement

IRCE_CALL_FUNC : CALL This token is used for calling the function

DESCRIPTION OF IRCE LANGUAGE

The starting point of our IRCE language is defined by "START" and the ending point is defined by "END". Programming any other way will give a syntax error. Though IRCE doesn't have a mandatory main method so with this characteristic, our language is different from many other programming languages. IRCE uses "=", "+", "-", "*", "/" signs to do operations. This helps with the writability and readability because any user with the basic math knowledge can do these operations and read them very easily. Again just like we mentioned our language also uses "!=", "==", ">", "<", ">=", "<=" comparator symbols so the user can read and write the code easily. IRCE to assign a value into a variable we use "IS" this makes the assignment statement very easy to be understood and to be written this again increases our languages readability and writability because we do not use equal signs so even if the value is a string we do not have confusion about the values type because every type is assigned with "IS". To assign a value the user needs to first specify the type of the variable then the name of the variable then "IS" and finally the value to be assigned.

Longitude is made of three integer values -which integers are numbers from zero to infinity- that has a space in between of the numbers. Latitude is same as the longitude; it is made of three integers. Address is made of strings which is all the numbers and all the upper and lower case letters.

Road can be established with integers or strings. User is estabilshed with variables which consist of single lower case letter or a lower case string. Crossroad can be made of integers or strings.

While assigning a value if the type doesn't match up with the type's limitations then the program will give a syntax error. Also, we didn't let the user add numbers to variables because we thought that it would confuse the programmer because we already made strings for that purpose which can contain both integers and letters. When we are opening up an array it starts with ARRAY the type of array which can be STR or INTEGER. Then the name of the array then our assignments statement which is "IS" , and finally we open up parentheses and in the inside we put the array's variables

which should match the array type otherwise there would be a syntax error and these variables are separated by empty spaces this makes the writability much easier because the programmer uses the keyboard more easily but at the same time if the programmer touched the space button by accident there would be more variables then intended. To open up a Graph the programmer first needs to add graphs type, which is most likely longitude then latitude. Then we need to add the name of the graph after that like always we add "IS" which is our assignment statement after all of this the programmer opens up parentheses but the difference from the array is that there can be multiple parentheses and inside these parentheses there should be values that wouldn't oppose to the graphs type. If statement starts with "IF" and after this we need to add a logical operation which are used with logical operators then in our program the programmer needs to add colon after this there should be a statement.

Statements can be assignment statement, array statement, graph statement, if statement, while statement, function define statement, function call statement, return statement or an expression. There doesn't have to be an else statement but if there is then we just need to write "ELSE" and then another statement inside. Also, in our program the statements always end with a semicolon. If statements whether it has else or not always ends with "ENDIF" after this we need to add a semicolon. Our if else statement because in if we add a colon but in else, we don’t add one this may be a little confusing to the programmer.

While statements start with "WHILE" token and then we need to add a logical expression then a colon and then statement lists after all of these to end the wile we use "ENDWHILE" token then we need to add a semicolon. Using tokens like "ENDIF" or "ENDWHILE" may decrease writability, it increases the readability because the programmer doesn’t count the "{}" parentheses.

For statements starts with "FOR" token then an assignment statement which is done with "IS" token then we add a comma then a logic expression and afterwards another comma and another assignment statement then a colon then a statement. And at the end we add "ENDFOR" token and add a semicolon to stop it.

To define a function we use function definition statement. This statement opens up with "FUNC" token. We open up parentheses inside these parentheses there are parameter lists and after we close the parentheses there is colons then statement lists, and like we mentioned they can be assignment statements, array statements, graph statement, if statements, while statements, function define statements, function call statements, return statements or an expressions. After everything the has to put ”ENDFUNC” and a semicolon to finish the declaration of a new function.

To call a built in or a defined function, we first write “CALL” and then we write the name of the function. After this we open up parentheses and inside we put the

parameters that we want to compute. Inside these parameters there should be as many parameters as the definition otherwise the program would give syntax error. Also we separate these parameters with empty spaces. At the end we add a semicolon.

IRCE language has eight built in function inside it and these are ShowOnMap, SearchLocation, GetRoadSpeed, GetLocation, ShowTarget, ShowCrossRoad, Print, FindDestination.

ShowOnMap function is called firstly by “CALL” then the name of function and this function takes longitude and latitude as parameters these parameters are separated by a comma and this function returns the location on the map.

SearchLocation function is called firstly by “CALL” then the name of function and this function takes address as a parameter which can be integers or letters. This function takes one parameter and otherwise given would give a syntax error. This function searches for the given address.

GetRoadSpeed function is called firstly by “CALL” then the name of function and this function takes road as the only parameter and road can be string or integer. This function returns the speed of the given road.

GetLocation like the other ones is called and then this function takes user as the only parameter which is lowercase letters then returns the location of the user.

ShowTarget is called and then takes address as the parameter. Then shows the specified address' location on map.

ShowCrossRoad is called then takes road as parameter. It returns true if road is crossroad and false if road is not crossroad.

Print is called and takes either string or a variable or integer and then prints it.

FindDestination is first called with “CALL”. This token takes two arguments which are longitude and latitude. These parameters are separated by a comma.It finds the destination location on map.

Overall IRCE is a practical language for GPS systems because the built in functions takes care of most of the problems also language’s writability and readability is increased by unique features in the language.

BNF GRAMMAR OF IRCE LANGUAGE

-- Built In Functions --

ShowOnMap()

SearchLocation()

GetRoadSpeed()

GetLocation()

ShowTarget()

ShowCrossRoad()

Print()

FindDestination()

--

<program> ::= <token_start><statement_list><token_end>

<token_start> ::= START

<token_end> ::= END

<statement_list> ::= <statement><SEMICOLON>

| <statement><SEMICOLON><statement_list>

<statement> ::= <assignment_statement>

| <array_statement>

| <graph_statement>

| <if_statement>

| <for_statement>

| <while_statement>

| <function_define_statement>

| <function_call_statement>

| <return_statement>

| <expression>

<assignment_statement> ::= <type><VARIABLE><TOKEN_IS><expression>

|

<type><VARIABLE><TOKEN_IS><function_call_statement>

| <type><VARIABLE><TOKEN_IS><logic_expression>

<array_statement> ::=

<ARRAY_TYPE><type><VARIABLE><TOKEN_IS><LEFT_PAR><array_elements><RIGHT_PAR>

<if_statement> ::=

<TOKEN_IF><logic_expression><COLON><statement_list><TOKEN_ELSE><statement_list><TOKEN_ENDIF

|

<TOKEN_IF><logic_expression><COLON><statement_list><TOKEN_ENDIF>

<for_statement> ::=

<TOKEN_FOR><assignment_statement><COMMA><logic_expression><COMMA><assignment_statement><COLON><statement_list><TOKEN_END_FOR>

<while_statement> ::=

<TOKEN_WHILE><logic_expression><COLON><statement_list><TOKEN_END_WHILE>

<logic_expression> ::= <term><logic_operator><term>

| <BOOLEAN>

<function_call_statement> ::=

<IRCE_CALL_FUNC><IDENTIFIER><LEFT_PAR><parameter_list><RIGHT_PAR>

|

<IRCE_CALL_FUNC><IDENTIFIER><LEFT_PAR><RIGHT_PAR>

|

<IRCE_CALL_FUNC><BLTIN_SHOW_ON_MAP><LEFT_PAR><longitude><COMMA><latitude><RIGHT_PAR>

|

<IRCE_CALL_FUNC><BLTIN_SEARCH_LOCATION><LEFT_PAR><address><RIGHT_PAR>

|

<IRCE_CALL_FUNC><BLTIN_GET_ROAD_SPEED><LEFT_PAR><road><RIGHT_PAR>

|

<IRCE_CALL_FUNC><BLTIN_GET_LOCATION><LEFT_PAR><user><RIGHT_PAR>

|

<IRCE_CALL_FUNC><BLTIN_SHOW_TARGET><LEFT_PAR><address><RIGHT_PAR>

|

<IRCE_CALL_FUNC><BLTIN_SHOW_CROSS_ROAD><LEFT_PAR><crossroad><RIGHT_PAR>

|

<IRCE_CALL_FUNC><BLTIN_PRINT><LEFT_PAR><term><RIGHT_PAR>

<define_parameter_list> ::= <type><term>

| <type><term><define_parameter_list>

<parameter_list> ::= <term>

| <term><parameter_list>

<type> ::= <INT_TYPE>

| <STRING_TYPE>

| <IRCE_ADDRESS>

| <IRCE_LATITUDE>

| <IRCE_LONGITUDE>

| <IRCE_ROAD>

| <IRCE_USER>

| <IRCE_CROSSROAD>

<function_define_statement> ::=
<IRCE_FUNC><IDENTIFIER><LEFT_PAR><define_parameter_list><RIGHT_PAR><COLON><statement_list><IRCE_END_FUNC>

|

<IRCE_FUNC><IDENTIFIER><LEFT_PAR><RIGHT_PAR><COLON><statement_list><IRCE_END_FUNC>

<return_statement> ::= <TOKEN_RETURN><expression>

| <TOKEN_NOTHING>

<expression> ::= <term><operator><expression>

| <term>

<operator> ::= +

| -

| *

| /

| =

<logic_operator> ::= >

| >=

| <

| <=

| != |
| == |
| AND |
| OR |
| NOT |

<longitude> ::= <INTEGER><INTEGER><INTEGER>

<latitude> ::= <INTEGER><INTEGER><INTEGER>

<address> ::= <STRING>

<road> ::= <STRING>
| <INTEGER>

<user> ::= <VARIABLE>

<crossroad> ::= <STRING>
| <INTEGER>

<lowercase_letter> ::= a | b | c | ç | d | e | f | g | ğ | h | ı | i | j | k | l | m | n | o | ö | p | q | r | s | ş | t | u | ü | v | w | x | y | z

<uppercase_letter> ::= A | B | C | Ç | D | E | F | G | Ğ | H | I | İ | J | K | L | M | N | O | Ö | P | Q | R | S | Ş | T | U | Ü | V | W | X | Y | Z

<VARIABLE> ::= <lowercase_letter>
| <lowercase_letter><VARIABLE>

<IDENTIFIER> ::= <uppercase_letter><IDENTIFIER>

| <IDENTIFIER><VARIABLE>

| <uppercase_letter><VARIABLE>

| <uppercase_letter><lowercase_letter>

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

<INTEGER> ::= <digit>

| <digit><INTEGER>

<INT_TYPE> ::= INT

<STRING_TYPE> ::= STR

<ARRAY_TYPE> ::= ARRAY

<GRAPH_TYPE> ::= GRAPH

<IRCE_ADDRESS> ::= ADDRESS

<IRCE_ROAD> ::= ROAD

<IRCE_USER> ::= USER

<IRCE_LONGITUDE> ::= LONGITUDE

<IRCE_LATITUDE> ::= LATITUDE

<IRCE_CROSSROAD> ::= CROSSROAD

<IRCE_FUNC> ::= FUNC

<IRCE_END_FUNC> ::= ENDFUNC

<IRCE_CALL_FUNC> ::= CALL

<BOOLEAN> ::= TRUE

| FALSE

<BLTIN_SHOW_ON_MAP>	::=	ShowOnMap
<BLTIN_SEARCH_LOCATION>	::=	SearchLocation
<BLTIN_GET_ROAD_SPEED>	::=	GetRoadSpeed
<BLTIN_GET_LOCATION>	::=	GetLocation
<BLTIN_SHOW_TARGET>	::=	ShowTarget
<BLTIN_SHOW_CROSS_ROAD>	::=	ShowCrossRoad
<BLTIN_PRINT>	::=	Print
<BLTIN_FIND_DESTINATION>	::=	FindDestination

<TOKEN_IS>	::=	IS
<TOKEN_RETURN>	::=	RETURN
<TOKEN_IF>	::=	IF
<TOKEN_ELSE>	::=	ELSE
<TOKEN_ENDIF>	::=	ENDIF
<TOKEN_WHILE>	::=	WHILE
<TOKEN_END_WHILE>	::=	ENDWHILE
<TOKEN_FOR>	::=	FOR
<TOKEN_END_FOR>	::=	ENDFOR
<TOKEN_NOTHING>	::=	NOTHING

<SEMICOLON>	::=	;
<LEFT_PAR>	::=	(
<RIGHT_PAR>	::=	)
<COLON>	::=	:
<DOT>	::=	.
<COMMA>	::=	,

ADDITIONAL FUNCTIONALITY

In our IRCE language, we added new built-in functions in exception to the ones we were assigned to do. ShowCrossRoad() function takes one parameter which is crossroad. – crossroad can be integer or string.- It determines the parameter is cross road or not. It is similar to Boolean data type but it is valid for only crossroads. It calls by CALL ShowCrossRoad (parameter). Print() is another one of our extra built in fuction this function can take every number and string and prints them into the console. FindDestination() is our final extra function and it takes longitude and latitude to find the specified destination.

In addition our program takes turkish alphabetical characters as well as the english ones.

CHALLENGES

While writing our language, IRCE we stumbled upon ambiguity after looking into ways to solve it we found left hand rule which is a association rule. In IRCE programming language, we watched out the association rule. We care about the tokens that not repeat more than one.

In IRCE, indentation is not important like C language, We used semicolon ";" to understand the end of a statement. Except semicolon, we used END token in for, while, if, function definition statements to understand the end of a statement.

After making the necessary changes in the yacc file our program worked without any problems.