-REPORT-

Dilara İŞERİ 21783561

Nur Bengisu ÇAM 21627097

Tuğçe BABACAN 21626986

In this assignment, we implemented our programming language for geographical information systems application. In our programming language, there are functions for finding the best road, finding the location and such. We created the BNF, lex and yacc of our language.

- In our language there is objects and their 3D object definitions. We have a class called as "3dObject" and in this class there are attributes such as Height, Width and Depth. We can create an object from this class in our application.
- There is also a class called as "point". There are attributes such as Longitude and Latitude. We can create point objects to use in our application. We can create a point called Destination to indicate the desired location and Position to indicate the current position of the user.
- There is also a class called "road". There are Destination, Position and RoadName features of our class. These Destination and Position features indicates the GPS information. For the "road definition", we can create a road object.
- There is also a class called node. This class is used to create the graph structure. There is another structure called array

Here is our built-in functions to create the GIS application:

1. showOnMap(float longitude, float latitude) :

   This function takes 2 floats as longitude and latitude. From the given parameters, this function show the GPS location of the point, either current location or a destination location point.

2. searchLocation(point address):

   This function takes a point object as an address, returns the longitude and latitude attributes of the point.

3. getRoadSpeed(road r):

   This function takes a road object as r. All the road objects have the attributes Destination, Position and RoadName. These Destination and Position values are actually a point. Each point has 2 attributes and these are Longitude and Latitude. For the given road r, this function calculates the speed based on the Destination and Position

4. getLocation(int user):

   This function takes an integer value indicating the user_id which is created for each user by default and stored in the system. Returns the location of the user, as a point object since every point has attributes as Longitude and Latitude.

5. showTarget(point address):

   This function takes point object as an address and shows the point attributes Longitude and Latitude on map.

6. createGraph():

This functions creates a graph structure and stores it in the application, returns the node object Root node.

7. addNode(node Root, node newNode):

This function takes a graph's root node as Root and a new node to be added into the graph indicating by the Root as newNode.

8. getCrossRoads(array allRoads[ ]):

This function takes an array of all the roads in the system. Each road is an object of the road class. Since the road class has attributes called Destination, Position and RoadName where Destination and Position attributes are actually point objects, this function finds which road in the system are cross roads based on the Destination and Position Longitude and Latitude.

9. getRoadDistance (road r):

This function takes the road object r and since the road class has attributes as Destination, Position and RoadName, based on the Destination, Position points' Longitude and Latitude, calculates the distance of the road.

10. getRoadScore (road r):

This function takes a road object r, calls the getRoadDistance(road r) and getRoadSpeed(road r) functions. Based on the return values of these called functions, getRoadScore(road r) function calculates a score value for the road. If the score is high, the road is good to go and vice versa.

11. getLocationOfOtherUsers(array users[ ]):

This function takes an array of all users in the application. Returns the location of the desired user among all users

BNF

Arithmetic Operators:
<arithmetic operators> ::= + | - | * | / | ^

Conditional Operators:
<conditional operators> ::= <logical operators> | <comparison operators>

Logical Operators:
<logical operators> ::= && | ! | ||

Comparison Operators:
<comparison operators> ::= <| <= | >| >= | == | !=

<check strings> ::= ==| !=

<*digits*> ::= <*digit*> | <*digits*> <*digit*>
<*digit*> ::= 0 | <*non zero digit*>
<*non zero digit*> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<*floating*> ::= <*digits*> **.** <*digits*> | <*exponent part*> | <*digits*> <*exponent part*>|
<*exponent part*> ::= <*exponent indicator*> *integer*>
<*exponent indicator*> ::= e
<*integer*> ::= <*sign*>? <*digits*>
<number> ::= <integer> | <floating>
<*sign*> ::= + | -
<*null*> ::= **NULL**
<letters> ::= a| b| c| d| e| f| g| h| i| j| k| l| m| n| o| p| 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| R| S| T| U| W| V| X| Y| Z
<chars> ::= <digits> | <letters>
<special chars> ::= \n | \r | \t
<punctuation> ::= .| ?| ,| ‘| “| (|)| ;| :| !| {| }| [|]| %| $| #| -| *| =
<text> ::= <text> |<chars> | <punctuation>
<string> ::= “<text>”
<boolean expression> ::= <expression><conditional operators><expression> | <expression>
| <string><check strings><string> |
<expression> ::= <variable name> | <number>
<any statement> ::= <if statement> | <assignment statement> | <loop statement> | <arithmetic operations>

Variable Names:

<variable name> ::= <variable name> | <letters><chars> | <letters>

Assignment Statements:

<declaration statement> ::= <variable type> <variable name> ; | <variable type> <assignment statement>
<assignment statement> ::= <*left hand side*> <*assignment operator*> <*assignment*> ;
<*left hand side*> ::= <*variable name*> | <*array access*> | <graph access>
<assignment> ::= <integer> | <floating> | <null> | <string> | <variable name>|<arithmetic statement>
<*assignment operator*> ::= **= |+= | -=**
Arithmetic operations:
<arithmetic statement> ::= <arithmetic statement> + <arithmetic statement_2> | <arithmetic statement_2>
<arithmetic statement_2> ::= <arithmetic statement_2> - <arithmetic statement_3> | <arithmetic statement_3>
<arithmetic statement_3> ::= <arithmetic statement_3> * <arithmetic statement_4> | <arithmetic statement_4>
<arithmetic statement_4> ::= <arithmetic statement_4> / <arithmetic statement_5> | <arithmetic statement_5>
<arithmetic statement_5> ::= (<arithmetic statement>) | <integer> | <floating>

Loops:

<loop statement> ::= <while loop> | <for loop>

<while loop> ::= while (<boolean expression>) then <any statement>

<for loop> ::= for (<assignment statement> ; <boolean expression>; <assignment statement>) then <any statement>

<any statement> ::= <any statement> | <any nonif statement> | <if statement>

Conditional statements:
<if statement> ::= <matched> | <unmatched>
<matched> ::= if (<boolean expression>) then <matched> else <matched> | <any nonif statement>
<unmatched> ::= if (<boolean expression>) then <if statement> | if <boolean expression> then <matched> else <unmatched>
<any nonif statement> ::= <any nonif statement> | <assignment statement>| <loop statement> | <assignment statement> | <function call>
Function definitions:
<function definition> ::= function <return type><function name> (<parameter list>) ; | <built-in functions> ;
<function name> ::= <variable name>
<parameter list> ::= <parameter list><parameter> | <parameter> | ε
<parameter> ::= <variable type><variable name>| <structure type><variable name> | <integer> | <float> | <string>
<variable type> ::= str | int | float | node
<structure type> ::= array
<return type> ::= <variable type> | void
Built-in functions:
<built-in functions> ::= showOnMap(float longitude,float latitude) |
searchLocation(string address) |
getRoadSpeed(road road) |
getLocation(int user) |
showTarget(string address) |
createGraph() |
addNode(node Root, node newNode) |
getCrossRoads(array allRoads[ ]) |
getRoadDistance (road r) |
getRoadScore (road r) |
getLocationOfOtherUsers(array users[ ])
Array data type:
<*array declaration*> ::= <variable type><variable name> [ ];
<array access> ::= <variable name>[<index>]
<index> ::= <digits>

Graph data type:
<*graph declaration*> ::= graph<variable name> { };
<graph access> ::= <variable name>

Function calls:
<function call> ::=<function name> (<call parameter list>) ;
<function name> ::= <variable name>
<call parameter list> ::= <call parameter list><call parameter> | <call parameter> | ε
<call parameter> ::= <variable name> | <integer> | <float> | <string>

RULES

- There is str data type which is corresponding to strings, int data type corresponding to integers, float data type corresponding to floats and identifier data type to be used for variable names.
- str data type must be used between two quotation marks when it's testing. That language only gets the token as string if there's quotation marks.
- str data type can be any char or combination of chars.
- int data type can be signed or unsigned.
- Punctuation can not be included in identifier data type.
- Identifiers can not be only digits.
- Identifiers can not start with a digit.
- There are also different data types such as array, graph, node and road to be able to used in built-in functions.
- All the arithmetic operations, conditional statements and loops must be implemented by using whitespaces between all the tokens.
- Conditional statements are used for comparing if two variables are equal, not equal, less or more than each other, or not.
- In "any statement" BNF, all the statements are included to be used in anywhere.

Definition of Language Constructs:

- For "if statement", conditional statement part is written between left and right brackets and the statement which is providing the condition is stated after "then" keyword.
- For "for loop", all the statements are written between left and right brackets and between all the statements, there must be a semicolon. For the following any statements, "then" keyword must be used same as in if statement.
- For "while loop", conditional statement part is written between left and right brackets and the statement which is providing the condition is stated after "then" keyword exactly same as in if statement.
- Return type does not be stated while functions are defining.
- In function definitions, after "function" keyword there must be given the function name ,that can be only identifier, and after that declaration between left and right brackets all the parameters must be stated with their data types if there are.
- In function calls, call must be done by using that identifier and just the parameter names between left and right brackets but there must be also a semicolon at the end of the line.
- To define an array, write array keyword before the array name.
- To define a graph, write graph keyword before the graph name.

Challenges:

While designing our language we in struggled some of the parts. BNF part was the easiest thing because we already knew it. Lex part was not that hard but we got some errors that we could not understand. We asked help from our TA and lecturers. We finally managed to solve them. For the Lex part, to see whether we did correctly or not, we complied the Lex part first. After we successfully compiled the Lex file, we moved on to the Yacc part. Yacc part was the hardest but did not take too much time since we have already defined our BNF. We took advantage of designing the BNF in detail.

How To Run?

You can run this program in dev.cs.hacettepe.edu.tr device by simply typing "make" on the command line. The line with the error will be printed to the screen and program will be compiled.