Phase One

Lexical Analyzer Generator

Mayar El Mahdy – 4639.

El Zahraa Emara – 4558.

**Part One : Data Structures used :**

1-Stack : Used this data structure in order to help in implementing the NFA automata , by pushing the characters of the Regular expression and popping them when I find either of these cases :

* When the character read is ‘(‘ , It will pop the elements from the stack
* When the expression is all pushed in the stack , then pop the expression and start building the nodes.
* When the character is ‘)’ then pop the elements from the stack.



**Figure : Usage of Stack**

2- ArrayList : Used the array list to add strings into a list , this was used several times in the code like in the class LexicalRules when reading the input file I simple add the Line I read into an ArrayList to handle it.

3- 2D arrays : This helps in storing the transition table.

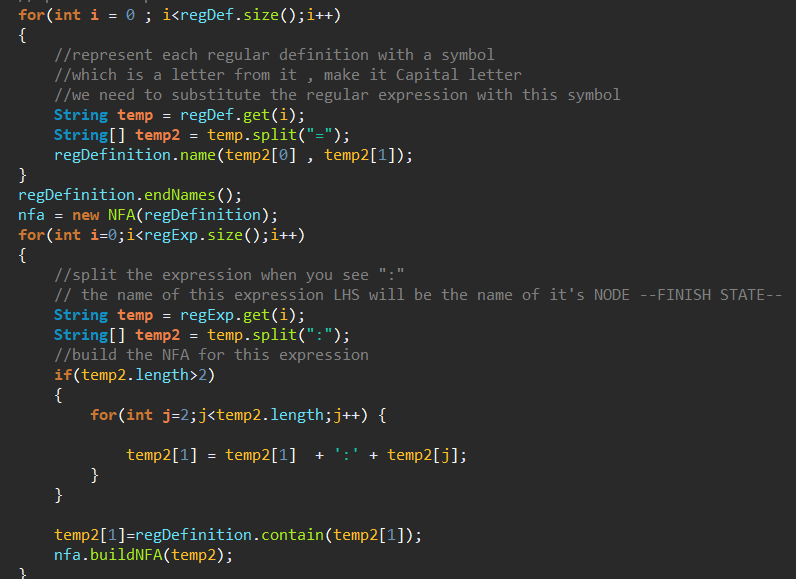
**Part two : Algorithms & Techniques used :**

1- Implemented an array of Nodes that was used in building the automata , It is a **graph** but implemented from scratch in order to add more functions to the nodes present.

2- The 2D array that was mentioned in the data structures section.

3- Split() , used this algorithm to split when I see the occurrence of a certain String

**For example :**

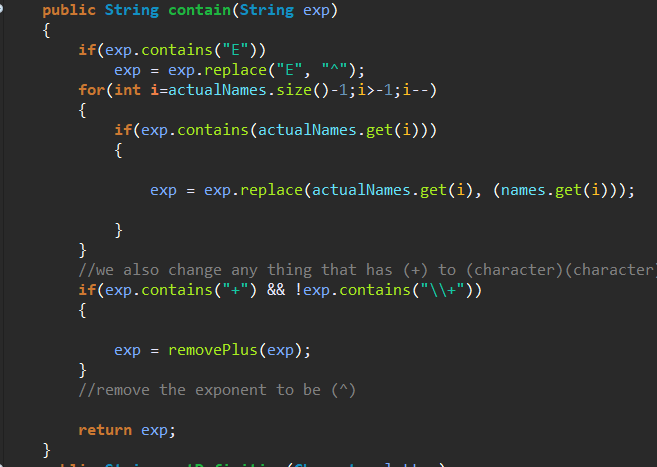


**Figure: Split()**

Here I split the regular expression and the regular definition whenever I find ( :) , (=).

4-replace() , Used this algorithm to replace a certain String with another one .

**For example:**



**Figure: Usage of replace()**

In the figure above , each regular expression containing a reference to a regular definition will be replaced by it’s alternative , also E will be (^).

**Part three : Transition Table for minimal DFA :**

**Part Four : Test files outputs :**

**1-Test Case Output 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case 1** | **Test Case 2** | **Test Case 3** | **Test from pdf** |
| program  id  ;  var  id  ,  id  :  integer  ;  begin  id  assign  num  ;  while  id  relop  num  do  begin  id  assign  id  addop  num  ;  read  (  id  )  ;  if  id  relop  num  then  id  assign  id  addop  num  else  id  assign  id  addop  id  end  ;  write  (  id  ,  id  )  end  . | program  id  ;  var  id  ,  id  :  integer  ;  begin  id  assign  num  ;  id  assign  floatNum  ;  id  assign  num  ;  while  id  relop  num  do  begin  id  assign  id  addop  num  ;  read  (  id  )  ;  if  id  relop  num  then  id  assign  id  addop  floatNum  else  id  assign  id  addop  id  end  ;  write  (  id  ,  id  )  end  . | program  id  ;  var  id  ,  id  :  integer  ;  begin  id  incop  ;  id  decop  ;  while  id  relop  num  do  begin  id  assign  id  addop  num  ;  read  (  id  )  ;  if  id  relop  num  then  id  assign  id  addop  floatNum  else  id  assign  id  addop  id  end  ;  write  (  id  ,  id  )  end  . | int  id  ,  id  ,  id  ,  id  ;  while  (  id  relop  num  )  {  id  assign  id  addop  num  ;  } |

**Part Five : Assumptions :**

* Assumed that the epsilon has a symbol (~)
* Assumed that whenever a ( E) was found in the regular expression then it will be replaced with the symbol (^) as the ( E) means exponent .
* Assumed that the regular definitions will be replaced with the capital letter of its first letter **ex**: letter = A-Z|a-z Would become : L = A-Z|a-z

So that it will be replaced when it is found in any regular expression **ex:**  id:letter\* would become id:L\*

* Assumed that the arrows would carry a value that is a character **always** , so when I read the regular expression if there are two character they would be separated as two different nodes
* The regular definitions will be replaced with an exception in every node

So that wouldn’t be any confusion if a node has several arrows

Making the regular expression L-{any other character that has an arrow in this node}.

* The code is sensitive to spaces for example in lexical rules.txt the punctuations should be separated by spaces

[\( \(]] 🡪 There should be a space between them.