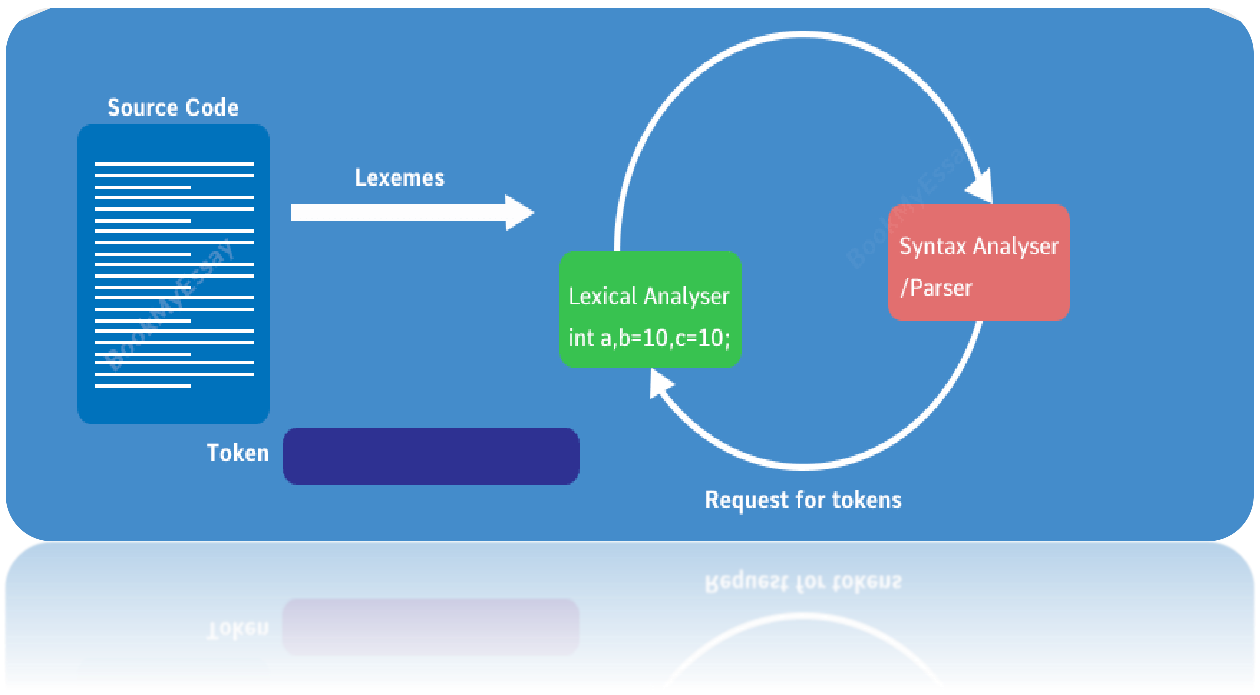


Compiler Project Phase 2

Parser

Due : 4/22/2020

Parser :

Second Step to develop a Compiler is to implement it's Parser. In this part of Project You are going to make a parser which will parse the language described in following pages and we would call it ***Roulang***. In order to accomplish this task you will be taught PGen and it's the only technology accepted from you for this part.

You can get the last edition of PGen from the link below :

https://github.com/Borjianamin98/PGen

Roulang Structures:

- ***Types***:

  Our language contains following types :

Type	Description
int	32 bit integer number
long	64 bit number
float	32 bit real number
double	64 bit real number
char	Any ASCII character
string	A sequence of characters
bool	Can be true or false
Self defined types	Programmar can define his own type by record keyword

- ***Variable:***

  Variable definition in ***Roulang*** is similar to java language. You can access any index of an array as a variable. Also you can access any field of a variable(in the case that it's a record) by following syntax :

  <variable name> . <field name>

- ***Literals***:

  Like every programming language , ***Roulang*** has some literals in addition to types, which can be assigned to variables.

- ***Expressions:***

  In ***Roulang***, expressions can appear in any of the following forms:

Expression	Example

A single variable	Int my_var = 3;
	My_var;

Casting **(<base_type>) <expression>**	(Long) my_var
Prefix/Postfix increment or decrement	My_var++ ; ++ My_var ; --My_var ; My_var-- ;
Any method call	Function() ; sizeof(<variable>)
Sizeof() : it gets a variable of a base type and returns it's size.	Sizeof(My_var);

- Obviously, any mathematical operation which involves two or more expressions as operands and following operators are also considered an expression:

Name	Symbol
addition	+
Subtraction / Unary Minus	-
multiplication	*
division	/
Modulo	%
Bitwise and	&
Bitwise or	|
Bitwise xor	^

Example:

- a * b + c / d
- 0x34 * 298 – c
- c & 0xFF00
- ….

- ***Roulang***, like most languages, provides Boolean Expressions in following syntax:
  - <expression> Comparison Operator <expression>
  - Boolean Unary Operator <Boolean expression>
  - <Boolean expression > Boolean Binary Operator <Boolean expression >

Name	Symbol	Type
Greater than	>	Comparison Operators
Smaller than	<	
Greater than Equal	>=	
Smaller than Equal	<=	
Equal	==	
unequal	!=	
And	and	Boolean Binary Operators
Or	or	
Exclusive Or	xor	
Not	not	Boolean Unary Operator

- ***Statements*** :
  - Condition:
    - If then Else:

      if (<Boolean expression> ***)*** <block> ***else*** <block>

      if (<Boolean expression> ***)*** <block>

        - Switch case :

          Switch (<expression>) **of**:

            begin

              case int_const : <block>

              ...

              default: <block>

            end

    - Loops:
        - For:

for (*** <assigment> ***; <condition-expression> ***;*** <step-expression> ***)*** <block>

            - Note that both assignment part and step-expression part are optional.

        - Repeat:

            repeat <block> ***until (*** <expression> ***) ;***

        - Foreach:

            foreach (id in <variable> ***)*** <block>

            - Note that in this loop variable must be iterable.

- Break and continue keywords can be used in body of all these loops.

- ***Assignment***:
  Assigning a value to a variable is done by one of the operators among : = , += , *= , -= , /= , %=

- ***Functions*** :
  Functions are defined as follow:

  function <type> ***id (*** <arguments> ***)*** <block>

  - A functions can have several arguments which are separated by comma ","
  - <type> can also be an array of a type.
  - Like C++ language Signature of a function can be declared but implement the body after the ***start*** function.
  - Each function must return an expression as return value except for void functions (in this case function ends with **return;**).
  - There are two special functions :
    - Println() : gets a String , number or variable as input and prints it with a line feed.
    - Input(<type>) : reads a value of the given type and returns it. If no type is given, reads and returns a string.

- ***Variable Declaration*** :
  Variables are declared by following syntax:
  <type> <brackets> <vars>;

  - <brackets> is used in the case that we want to declare an array
  - before <type> we can use **const** keyword to declare a constant variable. Constant variables must be assigned a value at the moment of declaration otherwise, a compile error must happen.
  - <vars> are name of variables that can be separated by comma.

- There are two special cases :
  - ***auto*** <vars>
    - in this case like constant variables all variables must be assigned a value at the moment of declaration otherwise, a compile error must happen. Example:

      ```
      auto a = 1, b = 2f, c = new int[1];
      ```
  - ***const auto*** <vars>:
    - everything is similar to previous case just it's not possible for <vars> to be an array and must result in a compile error.

- ***Record Declaration*:**

  Programmer can define his own type by the use of record keyword:

  ```
  record id {
          <var_dcl>
          <var_dcl>
          ...
  };
  ```

- ***Block*:**

  Blocks are used in loops , functions ,… :

  ```
  {
          <block_body>
  }
  ```

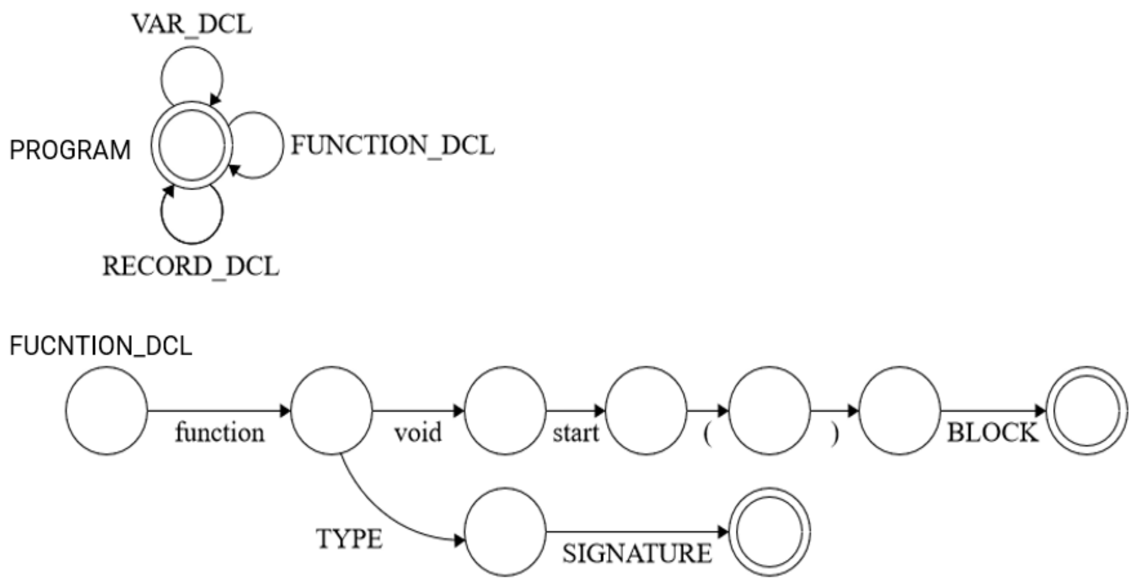
  - <block_body> can be variable declaration , statements , function calls,… or any combination of them.
  - Note that <block_body> can be empty.

- ***Program*:**

  The program is combination of variable declarations, function declarations and record declarations ***in any order.***

  Any program ***must*** contain a ***start*** function that has no input argument and it’s return value is void.

- ***Part of Parser Graph :***



- ***Notes***:
  - You may need to make some changes in your Scanner program so that it can pass tokens to your parser.
  - What you must upload is a zip file containing only a ***.pgs*** file (your diagrams) and a ***.prt*** file (your parse table).
  - This part of Project can be done in groups of two.

GOOD LUCK