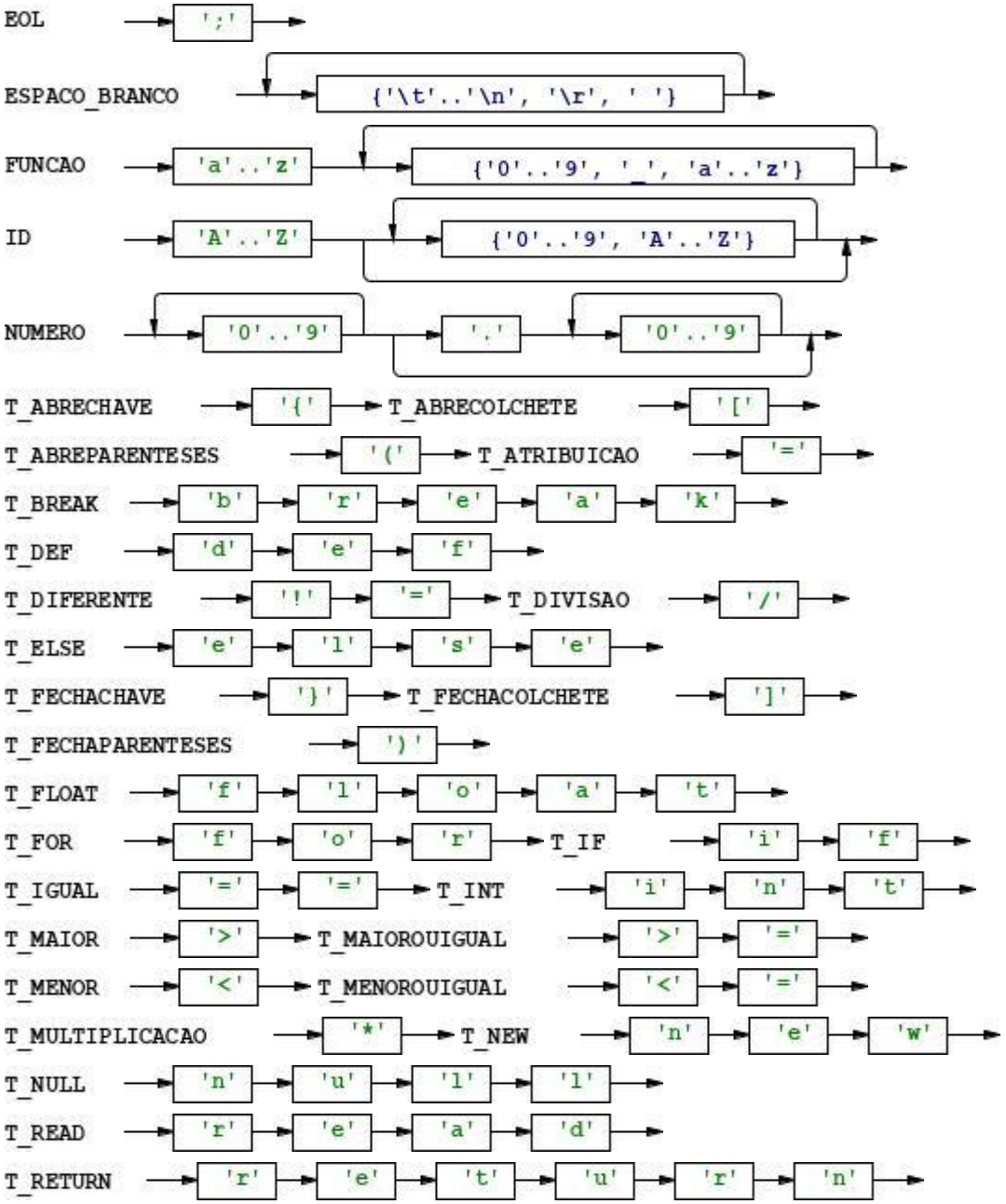
Lucas Tavares de Souza (16101113)
Elizeu Santos Madeira (20200610)

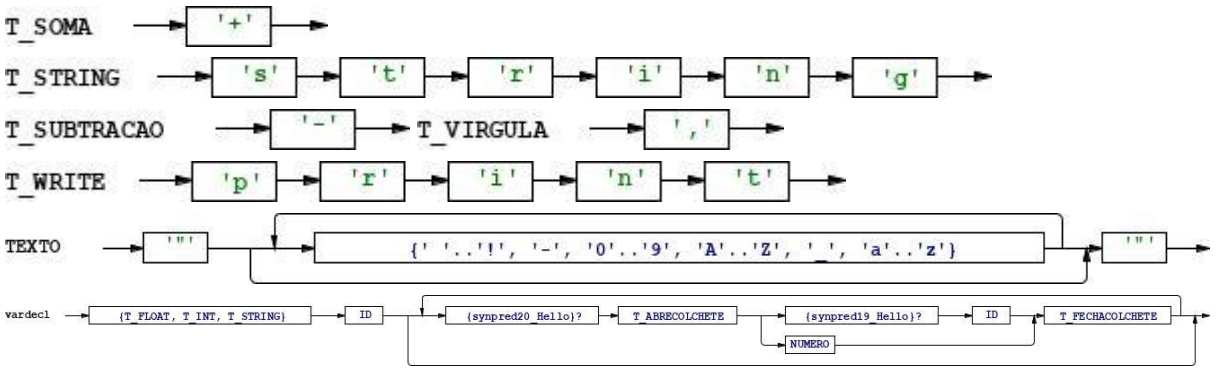
Tarefas AL

Questão 1: Produção das definições regulares para cada token;

arquivos .g

Questão 2: Construção dos diagramas de transição para cada token;





Questão 3: Descrição de uma tabela de símbolos (a tabela deverá guardar tokens identificadores seguido por pelo menos um atributo do token guardado na tabela);

Ao testar o programa, o sistema gerará um arquivo txt com o conteúdo da tabela de tokens. O nome do arquivo é "Tabela de tokens " + <nome do programa> + ".txt". Estamos enviando a tabela de tokens dos programas de teste exigidos no trabalho, sendo eles "**Tabela de tokens programa.lcc.txt**", "**Tabela de tokens programa2.lcc.txt**", "**Tabela de tokens programa3.lcc.txt**" e "**Tabela de tokens test.lcc.txt**"

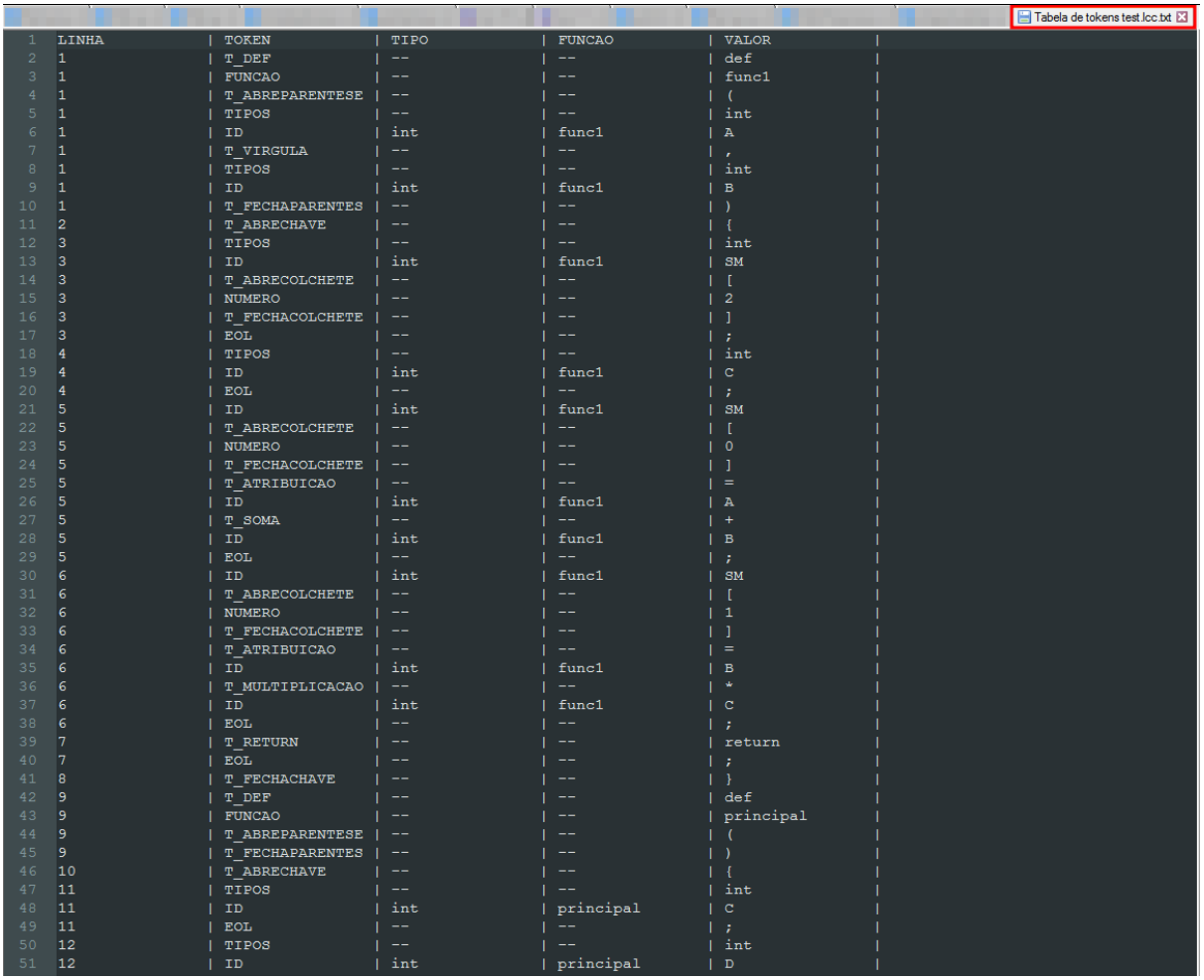


Tabela de tokens test.lcc.txt

LINHA	TOKEN	TIPO	FUNCAO	VALOR
1	T_DEF	--	--	def
1	FUNCAO	--	--	func1
1	T_ABREPARENTESE	--	--	(
1	TIPOS	--	--	int
1	ID	int	func1	A
1	T_VIRGULA	--	--	,
1	TIPOS	--	--	int
1	ID	int	func1	B
1	T_FECHAPARENTES	--	--	)
2	T_ABRECHAVE	--	--	{
3	TIPOS	--	--	int
3	ID	int	func1	SM
3	T_ABRECOLCHETE	--	--	[
3	NUMERO	--	--	2
3	T_FECHACOLCHETE	--	--	]
3	EOL	--	--	;
4	TIPOS	--	--	int
4	ID	int	func1	C
4	EOL	--	--	;
5	ID	int	func1	SM
5	T_ABRECOLCHETE	--	--	[
5	NUMERO	--	--	0
5	T_FECHACOLCHETE	--	--	]
5	T_ATRIBUICAO	--	--	=
5	ID	int	func1	A
5	T_SOMA	--	--	+
5	ID	int	func1	B
5	EOL	--	--	;
6	ID	int	func1	SM
6	T_ABRECOLCHETE	--	--	[
6	NUMERO	--	--	1
6	T_FECHACOLCHETE	--	--	]
6	T_ATRIBUICAO	--	--	=
6	ID	int	func1	B
6	T_MULTIPLICACAO	--	--	*
6	ID	int	func1	C
6	EOL	--	--	;
7	T_RETURN	--	--	return
7	EOL	--	--	;
8	T_FECHACHAVE	--	--	}
9	T_DEF	--	--	def
9	FUNCAO	--	--	principal
9	T_ABREPARENTESE	--	--	(
9	T_FECHAPARENTES	--	--	)
10	T_ABRECHAVE	--	--	{
11	TIPOS	--	--	int
11	ID	int	principal	C
11	EOL	--	--	;
12	TIPOS	--	--	int
12	ID	int	principal	D

Questão 4: Se não usou ferramenta, uma descrição da implementação do analisador léxico (Usou diagramas de transição? Quais? Quantos? Se não usou diagramas de transição, então o que foi usado?);

Questão 5: Se usou ferramenta, uma descrição detalhada da entrada exigida pela ferramenta e da saída dada por ela. É necessário haver exemplos pequenos da entrada e da saída gerada pela ferramenta com essa entrada.

Nesse trabalho foi usada a ferramenta Antlr v3.5.2. A gramática é escrita em um arquivo com extensão “.g” onde as regras Léxicas e Sintáticas ficam armazenadas (é possível separar em múltiplos arquivos para facilitar a organização). Segue um exemplo pequeno de uma gramática que reconhece expressões matemáticas:

```
grammar SimpleCalc;

tokens {
    PLUS    = '+' ;
    MINUS   = '-' ;
    MULT    = '*' ;
    DIV = '/' ;
}

@members {
    public static void main(String[] args) throws Exception {
        SimpleCalcLexer lex = new SimpleCalcLexer(new ANTLRFileStream(args[0]));
        CommonTokenStream tokens = new CommonTokenStream(lex);

        SimpleCalcParser parser = new SimpleCalcParser(tokens);

        try {
            parser.expr();
        } catch (RecognitionException e)  {
            e.printStackTrace();
        }
    }
}

/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/

expr    : term ( ( PLUS | MINUS )  term )* ;

term    : factor ( ( MULT | DIV ) factor )* ;

factor  : NUMBER ;


/*------------------------------------------------------------------
 * LEXER RULES
 *------------------------------------------------------------------*/

NUMBER  : (DIGIT)+ ;

WHITESPACE : ( '\t' | ' ' | '\r' | '\n'| '\u000C' )+    { $channel = HIDDEN; } ;

fragment DIGIT  : '0'..'9' ;
```

A ferramenta então exporta a gramática para classes java. O nome das classes são:

- SimpleCalcLexer: exporta as regras léxicas da gramática;
- SimpleCalcParser: exporta as regras sintáticas da gramática;

Com as classes exportadas, basta criar compilar usando comandos do java. É possível adicionar outras classes java para auxiliar com as regras léxicas/sintáticas ou quaisquer outras atividades que envolvam o projeto.

Mais detalhes sobre compilação e execução de exemplos no readme do projeto.

Tarefas AS

Questão 1: CC-2021-2 está na forma BNF. Coloque-a na forma convencional de gramática. Chame tal gramática de LCC-2021-2.

Arquivo gramatica_convencional.txt possui a gramática na forma convencional

```
PROGRAM          ->      STATEMENT | FUNCLIST
FUNCLIST         ->  FUNCDEF FUNCLIST_LINHA
FUNCLIST_LINHA   ->      FUNCLIST | ε
FUNCDEF          ->      'def' idfuncao '(' PARAMLIST ')' '{' STATELIST '}'
PARAMLIST        ->  TIPOS id PARAMLIST_LINHA  | ε
PARAMLIST_LINHA ->       ',' PARAMLIST | ε
STATELIST        ->  STATEMENT STATELIST_LINHA
STATELIST_LINHA ->       STATELIST | ε
STATEMENT        ->  VARDECL ',' | ATRIBSTAT ',' | PRINTSTAT ',' | READSTAT ',' | RETURNSTAT ',' | 'break' ',' | ',' | IFSTAT | FORSTAT | '{' STATELIST '}'
VARDECL          ->      TIPOS id VARDECL_LINHA
VARDECL_LINHA    ->      '['  numero ']' VARDECL_LINHA
LVALUE           ->      id LVALUE_LINHA
LVALUE_LINHA     ->      '[' NUMEXPRESSION ']' LVALUE_LINHA  | ε
ATRIBSTAT        ->  LVALUE '=' ATRIBSTAT_LINHA
ATRIBSTAT_LINHA ->       EXPRESSION | ALLOCEXPRESSION | FUNCCALL | texto
ALLOCEXPRESSION ->        'new' TIPOS ALLOCEXPRESSION_LINHA
ALLOCEXPRESSION_LINHA    ->      '[' NUMEXPRESSION ']' ALLOCEXPRESSION_LINHA
FUNCCALL         ->  idfuncao '(' PARAMLISTCALL ')'
PARAMLISTCALL    ->      texto PARAMLISTCALL_LINHA | EXPRESSION PARAMLISTCALL_LINHA | ε
PARAMLISTCALL_LINHA          ->  ',' PARAMLISTCALL | ε
PRINTSTAT        ->  'print' PRINTSTAT_LINHA
PRINTSTAT_LINHA ->       texto | EXPRESSION
READSTAT         ->  'read' LVALUE
RETURNSTAT       ->  'return' RETURNSTAT_LINHA
RETURNSTAT_LINHA->       texto | EXPRESSION | ε
IFSTAT           ->      'if' '(' EXPRESSION ')' IFSTAT_LINHA
IFSTAT_LINHA     ->      'else' STATEMENT | ε
FORSTAT          ->      'for' '(' ATRIBSTAT ';' EXPRESSION ';' ATRIBSTAT ')'  STATEMENT
EXPRESSION       ->  NUMEXPRESSION EXPRESSION_LINHA
EXPRESSION_LINHA->       '>' NUMEXPRESSION | '<' NUMEXPRESSION | '>=' NUMEXPRESSION | '<=' NUMEXPRESSION | '==' NUMEXPRESSION | '!=' NUMEXPRESSION | ε
NUMEXPRESSION    ->      TERM NUMEXPRESSION_LINHA
NUMEXPRESSION_LINHA      ->      '+' TERM NUMEXPRESSION_LINHA | '-' TERM NUMEXPRESSION_LINHA | ε
TERM             ->      UNARYEXPR TERM_LINHA
TERM_LINHA       ->  '*' UNARYEXPR | '/' UNARYEXPR | ε
UNARYEXPR        ->  '+' FACTOR | '-' FACTOR | FACTOR
FACTOR           ->      numero | LVALUE  | 'null' | '(' NUMEXPRESSION ')'
TIPOS            -> 'int' | 'float' | 'string'
```

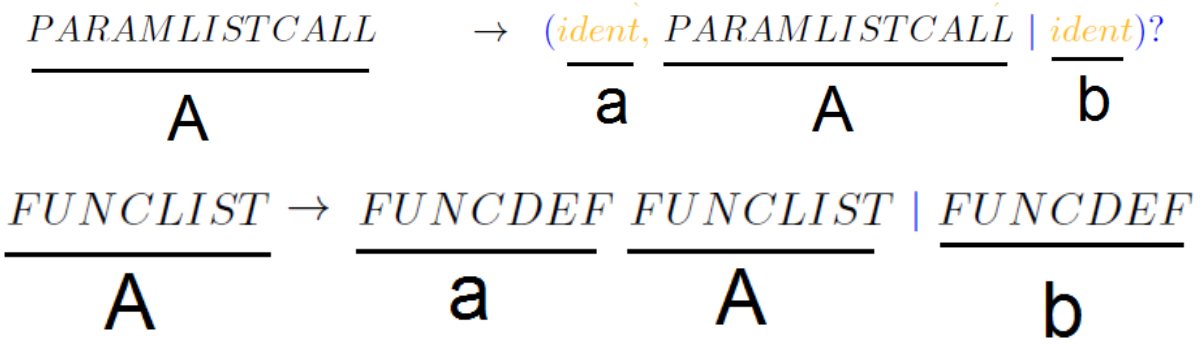
Questão 2: A sua LCC-2021-2 possui recursão à esquerda? Justifique detalhadamente sua resposta. Se ela tiver recursão à esquerda, então remova tal recursão.

A nossa LCC-2021-2 não possui recursão à esquerda, pois todas as regras jamais aparecem à esquerda de suas produções. Ou seja, sempre que há uma produção recursiva, primeiro vem um valor terminal ou não-terminal antes da recursão como é o caso da produção “statelist”.

```
279
280   statelist : statement (statelist)?
281       ;
282
```

Considerando a gramática em sua forma convencional (como pode ser visto no arquivo “gramatica_convencional.txt”) as produções foram quebradas em diversas produções auxiliares, portanto nem recursão direta há mais.

A LLC-2021-2 original também não possui recursão à esquerda. As recursões que possuem, são à direita, como nos exemplos abaixo:



Onde o ‘A’ está à direita de ‘alpha’ (a), portanto com recursão à direita.

Questão 3: A sua LCC-2021-2 está fatorada à esquerda? Justifique detalhadamente sua resposta. Se ela não estiver fatorada à esquerda, então fatore.

A linguagem estava fatorada à esquerda na regra “paramlist” pois o início das produções coincidem (TIPOS ID) e o subsequente é opcional (T_VIRGULA paramlist):

```
285   paramlist
286       : (
287           (T_INT   | T_FLOAT | T_STRING) {adicionaToken(input.LT(1));} ID |
288           (T_INT   | T_FLOAT | T_STRING) {adicionaToken(input.LT(1));} ID  T_VIRGULA paramlist
289       )?
290       ;
```

Para arrumar a fatoração, criou-se a regra “paramlist_linha”

```
284
285  paramlist
286      : ( (T_INT  | T_FLOAT | T_STRING) {adicionaToken(input.LT(1));} ID paramlist_linha )?
287      ;
288
289  paramlist_linha
290      : T_VIRGULA paramlist
291      | /* epsilon */
292      ;
```

Desta forma sempre que o interpretador produzir “TIPOS ID” ele não precisará voltar na árvore de parseamento quando o programa não possuir “T_VIRGULA paramlist” e produzir somente a primeira parte pois agora a segunda parte da produção é uma produção separada (paramlist_linha) e pode dar continuidade ou não à lista de parametros.

Da mesma forma “paramlistcall”:

```
322  paramlistcall
323      :   (
324              ({verificaToken(input.LT(1));} ID | TEXTO | expression) T_VIRGULA paramlistcall |
325              ({verificaToken(input.LT(1));} ID | TEXTO | expression)
326          )?
327      ;
328
```

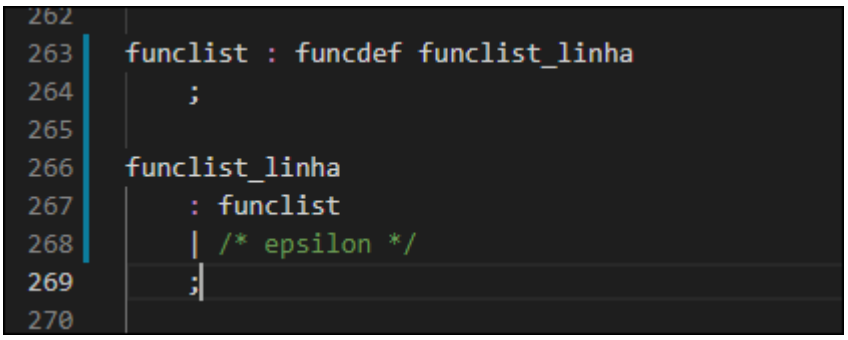
para:

```
322  paramlistcall
323      :   (
324              ({verificaToken(input.LT(1));} ID | TEXTO | expression) paramlistcall_linha
325          )?
326      ;
327
328  paramlistcall_linha
329      :
330      T_VIRGULA paramlistcall
331      | /* epsilon */
332      ;
```

Da mesma forma “funclist”:

```
261      ;
262
263  funclist : funcdef funclist |
264             funcdef
265      ;
266
```

para:



```
262
263 funclist : funcdef funclist_linha
264     ;
265
266 funclist_linha
267     : funclist
268     | /* epsilon */
269     ;
270
```

Questão 4: Faça LCC-2021-2 ser uma gramática em LL(1). É permitido adicionar novos terminais na gramática, se achar necessário. Depois disso, mostre que LCC-2021-2 está em LL(1) (você pode usar o Teorema ou a tabela de reconhecimento sintático vistos em videoaula).

A gramática é uma gramática LL(1). Originalmente obtivemos alguns problemas com mais produções que dava para o mesmo valor terminal (no caso, não estava em LL(1)), mas após algumas mudanças conseguimos ajustar. Basicamente, quando opcionalmente podíamos chamar, por exemplo “ID | TEXTO | EXPRESSION” dava problema, pois EXPRESSION gera ID (fazendo a gramática não estar em LL(1)). A tabela de reconhecimento sintático está junto do projeto em uma planilha chamada “first-follow.ods”

Questão 5: se não usou ferramenta, uma descrição da implementação do analisador sintático (Usou uma tabela de reconhecimento sintático para gramáticas em LL(1)? Se não, então o que foi usado?);

–

Questão 6: se usou ferramenta, uma descrição da entrada exigida pela ferramenta e da saída dada por ela.

Mudanças na linguagem

Mudanças na linguagem Algumas mudanças na linguagem solicitada pelo professor foram necessárias, muitas delas para ser possível implementar os exemplos de programas corretamente. A linguagem foi “aumentada” visto que a sugestão inicial definida no enunciado do trabalho é bastante limitada. As mudanças solicitada/sugeridas nos enunciados das questões de AS e AL foram comentadas e explicadas nas respostas do relatório. Abaixo estão apenas as mudanças implementadas fora do escopo das questões do trabalho. Entre as mudanças estão:

- Existe um ident foi separado entre ID e FUNCAO, pois pelo exemplo 1, a regra parece estar variáveis em maiúsculo e funções em minúsculos. Portanto separamos para podermos criar duas regras para ID e FUNCAO;
- As regras foram escritas com letras minúsculas e as os tokens em letras maiúsculas para ficarem na convenção da ferramenta usada (antlr);
- As variáveis possuem escopo de função (mas não de bloco). A ferramenta até implementa um escopo de bloco nativo, mas eu inseri apenas o de função para poder repetir nomes de variáveis em funções diferentes e facilitar a nomenclatura das variáveis nos exemplos;
- "atribstat" agora aceita textos como valor de atribuição, pois string é um tipo possível na nossa linguagem;
- "paramlistcall" agora aceita textos e expressões como parâmetro de envio. Nenhum motivo prático, apenas para ficar mais parecido com uma linguagem real;
- "printstat" e "returnstat" agora permite a escrita de textos ou expressões. Nenhum motivo prático, apenas para ficar mais parecido com uma linguagem real.