Centro Federal de Educação Tecnológica de Minas Gerais
Engenharia de Computação
Compiladores **Exercício 3**

RESPOSTAS

1) É possível implementar um parser recursivo descendente para a gramática a seguir? Explique.

S → SA | A
A → a

Resposta:
A gramática não é LL(1). Ela possui recursão à esquerda: S → SA | A. *FIRST (A) = {a} e FIRST(SA) = FIRST(A) = {a}. Desta forma, não é possível implementar um parser preditivo para essa gramática. Isso pode ser também observado na tabela do parser: as regras* S → SA e S → A *são incluídas na mesma entrada da tabela (linha S coluna a).*

	a	*$*
S	S → SA S → A	-
A	A → a	-

2) É possível implementar um parser recursivo descendente para a gramática a seguir? Explique.

S → A a A b | B b B a
A → λ
B → λ

Resposta:
A gramática é LL(1), os conjuntos FIRST das produções não são sobrepostos.
*Isso pode ser também observado construindo-se a tabela do parser preditivo. Na tabela, para um mesmo par (*não terminal, terminal*) não há entrada múltiplas. É possível, então, implementar um parser preditivo para a gramática.*

	FIRST	*FOLLOW*
S	*a,b*	*$*
A	λ	*a,b*
B	λ	*b,a*

	a	*B*
S	S → A a A b	S → B b B a
A	A → λ	A → λ
B	B → λ	B → λ

3) Considere a gramática a seguir

1. program → **begin** statementList **end**

2. statementList → statement statementTail
3. statementTail → statement statementTail
4. | λ
5. statement → id := expression;
6. | **read** (idList);
7. | **write** (exprList);
8. idList → id idTtail
9. idTail → , id idTail
10. | λ
11. exprList → expression exprTail
12. exprTail → , expression exprTail
13. | λ
14. expression → primary primaryTail
15. primaryTail → addOp primary primaryTail
16. | λ
17. primary → (expression)
18. | id
19. | intLiteral
20. addOp → +
21. | -

Na gramática, o símbolo id denota o padrão de formação de identificadores, e o símbolo intLiteral denota o padrão de formação de constantes numéricas inteiras.

a) Compute os conjuntos FIRST para os símbolos não terminais. (EM SALA)

b) Compute os conjuntos FOLLOW para os símbolos não terminais. (EM SALA)

Resposta:

Símbolo	**FIRST**	**FOLLOW**
program	begin	$
statementList	id, read, write	end
statementTail	id, read, write, λ	end
statement	id, read, write	id, read, write, end
idList	Id	)
idTail	“,” , λ	)
exprList	(, id, intLiteral	)
exprTail	“,” , λ	)
expression	(, id, intLiteral	; , “,” ,)
primaryTail	+,-,λ	; , “,” ,)
primary	(, id, intLiteral	+,-, ; , “,” ,)
addOp	+,-	(, id, intLiteral

c) Mostre a tabela de parser preditivo para a gramática. (EM SALA)

Resposta:

Obs.: as regras foram numeradas.

	begin	id	read	write	end	,	(	)	intLiteral	+	-	;
program	1											
statementList		2	2	2								
statementTail		3	3	3	4							
statement		5	6	7								
idList		8										
idTail						9		10				
exprList		11					11		11			
exprTail						12		13				
expression		14					14		14			
primaryTail						16		16		15	15	16
primary		18					17		19			
addOp										20	21	

d) Esta gramática é LL(1)? Justifique sua resposta.

Resposta: Sim, a gramátia é LL(1), pois não há par (não terminal, terminal) com entradas múltiplas na tabela do parser preditivo.

e) Mostre uma implementação do procedimento do parser preditivo para o símbolo *primaryTail.* (EM SALA)

Resposta:

```
void primaryTail(){
   switch(token){
        case MAIS:    // addOp primary primaryTail
        case MENOS: addOp();  primary(); primaryTail();
                          break;
        case VIRGULA: // λ
        case FP:
        case PVG: break;
        default: error();
     }

   }
```

f) Mostre uma implementação do procedimento do parser preditivo para o símbolo *statement*.

Resposta:

```
void statement(){
   switch(token){
        case ID:       // id := expression;
                        eat(ID); eat(ATRIB); expression(); eat(PVG);
                        break;
        case READ: // read (idList);
                       eat(READ); eat(AP); idList(); eat(FP); eat(PVG);
                        break;
        case WRITE: // write (exprList);
```

```
                    eat(WRITE); eat(AP); exprList(); eat(FP); eat(PVG);
                     break;
    default: error();
  }

}
```