CET087 – Conceitos de Linguagens de Programação – P2 – Quarta feira, dia 11 de maio de 2022

G5	Matrícula:	Nome:
	2019102821	Igor Lima Rocha

0,0 **1.-(2,0 pontos)**

a) Defina a notação BNF e a notação EBNF (BNF estendido).
b) Defina gramática livre de contexto ambígua.
c) Defina Gramáticas de Atributos.
d) Defina Semântica Operacional, Semântica denotacional e Semântica Axiomática.

1,5 **2.-(2,0 pontos)** Considerando a gramática, com símbolo inicial **<program>**:

p_0: **<program> → <stmts>**
p_1: **<stmts> → <stmt> | <stmt> ; <stmts>**
p_3: **<stmt> → <var> = <expr>**
p_4: **<var> → a | b | c | d**
p_8: **<expr> → <term> {(+ | -) <term>}**
p_{11}: **<term> → <factor> {(* | /) <factor>}**
p_{14}: **<factor> → <var> | 0 | 1 | (<expr>)**
~~p_{16}: **<term> → <var> | 0 | 1**~~

+/- a) Aplique as produções da gramática para obter a derivação de **a = (1+(0*1))**
+/- b) Desenhe a árvore de derivação correspondente à derivação encontrada em (2.a).

2,0 **3.-(2,0 pontos)** Considerando a gramática ambígua, com símbolo inicial **<expr>**:

p_0: **<expr> → <expr> <op> <expr> |** ~~**0 | 1**~~ const
p_2: **<op> → / | -**

Obtenha duas árvores de derivação para a palavra **const - const / const**

1,5 4.**(2,0 pontos)** Considerando a gramática, com símbolo inicial **<expr>**:

+/- p_0: **<expr> → <term> { (+ | -) <term>}**
+/- p_3: **<term> → <factor> { (* | /) <factor>}**
+/- p_6: **<factor> → x | y | z | 0 | 1 | (<expr>)**

Desenvolva na linguagem C o analisador descendente recursivo para essa gramática.

1,5 5.**(2,0 pontos)** Considerando a gramática, com símbolo inicial **<if_stmt>**:

+/- p_0: **<if_stmt> → if (<logic_expr>) { <stmt> }**
✓ p_1: **<stmt> → <var> = <expr>**
✓ p_2: **<var> → a | b | c | d**
+/- p_6: **<expr> → <term> { (+ | -) <term> }**
+/- p_9: **<term> → <factor> { (* | /) <factor> }**
+/- p_{12}: **<factor> → <var> | 0 | 1 | (<expr>)**
~~p_{16}: **<term> → <var> | 0 | 1**~~
+/- p_{19}: **<logic_expr> → F | T | p | q | (<logic_expr> <Y> <logic_expr>)**
✓ p_{24}: **<Y> → & | ! | < | >**

Desenvolva na linguagem C o analisador descendente recursivo para essa gramática.

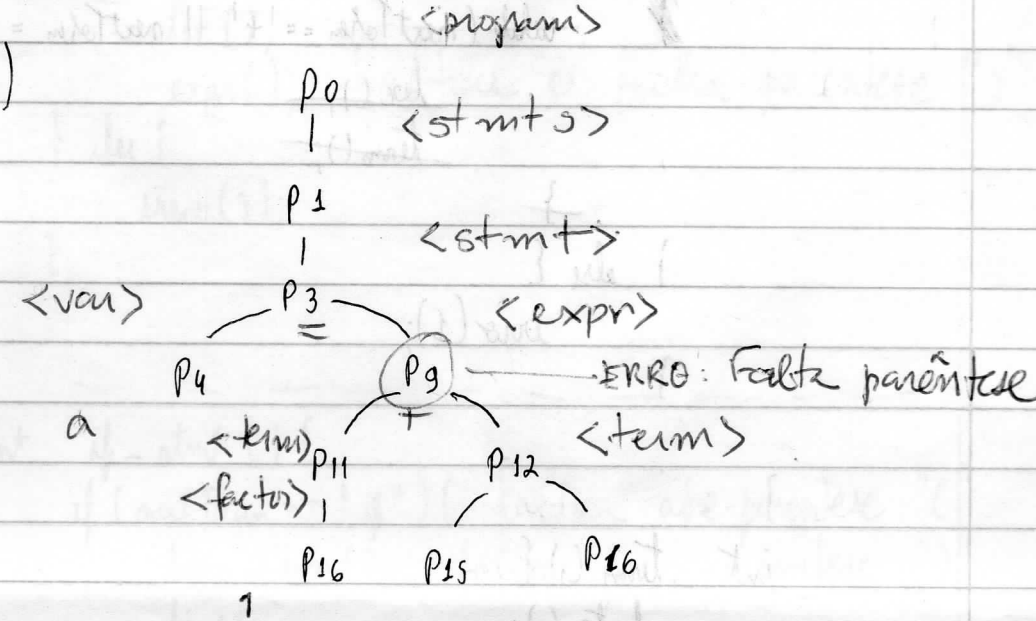
Igor Lima Rocha FOLHA 1 2019110282

2- a = (1 + (0 * 1))

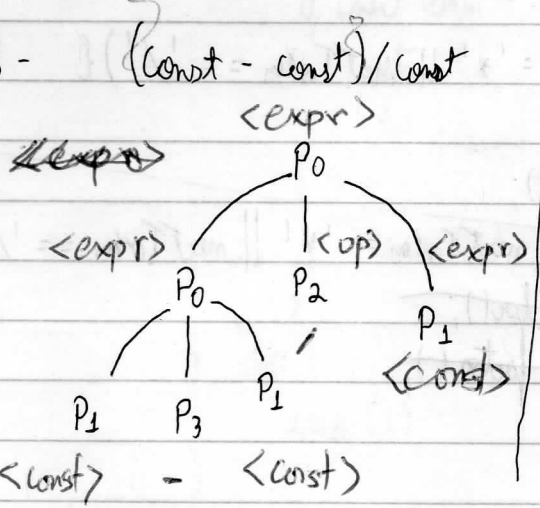
a) ~~P0 P1 P3 P4 P8 P11~~

P0 P1 P3 P4 P9 P11 P16 P12 P15 P16

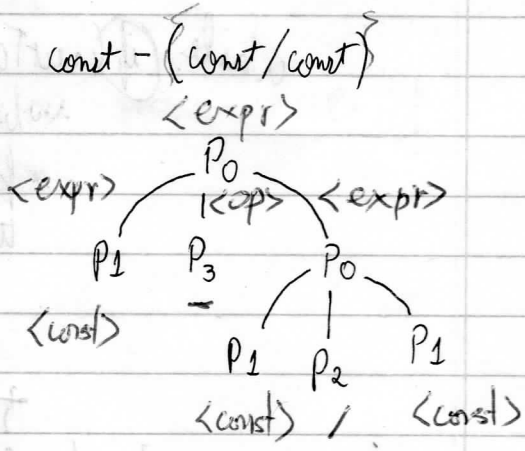
b)



3- (const - const) / const



const - (const / const)



4-

```
int expr() {
    term();
    while (nextToken == '+' || nextToken == '-') {
        lex();
        term();
        while (nextToken == '+' || nextToken == '-') {
            lex();
            term();
        }
    } else {
        erro(1);
    }
}
int term() {
    factor();
    while (nextToken == '*' || nextToken == '/') {
        lex();
        factor();
        while (nextToken == '*' || nextToken == '/') {
            lex();
            factor();
        }
    } else {
        erro(2)
    }
}
```

// próxima página

```
int factor () {
    if ( nextToken == 'x' || nextToken == 'y' ||
         nextToken == 'z' || nextToken == '0' ||
         nextToken == '1' ) {
        lex ();
        getc (nextTOKen);
    } else if (nextTOKen == '(') {
        lex();
        expr(); faltou o fecha-parentese ")"
    } else {
        erro (3);
    }
}
```

} o lex faz o getc (nextToken

5.

```
int if_stmt () {
    if (nextToken == "if") { faltou abe-parentese "("
        logic_expr (); faltou fecha-parentese ")"
        lex();
        if (nextTOKen == '{') {
            stmt ();
            lex ();  faltou fecha-chave "}"
        } else {  faltou
            erro (2);
        }
    } else {
        erro (1);
    }
}
int stmt () {
    var ();
    if (nextToken == '=') {
        expr ();
        lex();
    } else {
        erro (3);
    }
```

```
int var () {
    if ( nextToken == 'a' || nextToken == 'b' ||
         nextToken == 'c' || nextToken == 'd') {
        lex();
        getc (nextToken);
    } else {
        erro(4);
    }
}

int expr() // mesma função da questão 4
int term() // mesma função da questão 4
int factor() {
    ~~.....c);~~
    if ( nextToken == '0' || nextToken == '1') { }
        lex();
        getc (nextTOKen);
    } else if ( nextTOKen == '(') {
        expr();
        lex();
    } else {
        var();
        lex();
    }
}
int logic_expr () {
    if ( nextTOKen == 'F' || nextToken == 'T' ||
         nextTOKen == 'p' || nextToken == 'q') {
        lex();
        getc (nextToken);
    } else if ( nextTOKen == '(') {
        logic_expr();
        Y();
        logic_expr();
        lex();
    } else {
        erro(5);
    }
```

Faltou fecha-parentese ")"

Faltou fecha-parentese ")"

... continuação na folha 2

5-
```
...
int Y(){
    if (nextToken == '&' || nextToken == '!' ||
        nextToken == '<' || nextToken == '>'){
        lex();
        getc(nextTOKen);
    } else {
        erro(6);
    }
}
```