Relatório do Terceiro Laboratório de CES-41
Compiladores

3 de maio de 2018

Disciplina: CES-41

Estudante: Felipe Guimarães

Turma 19.3

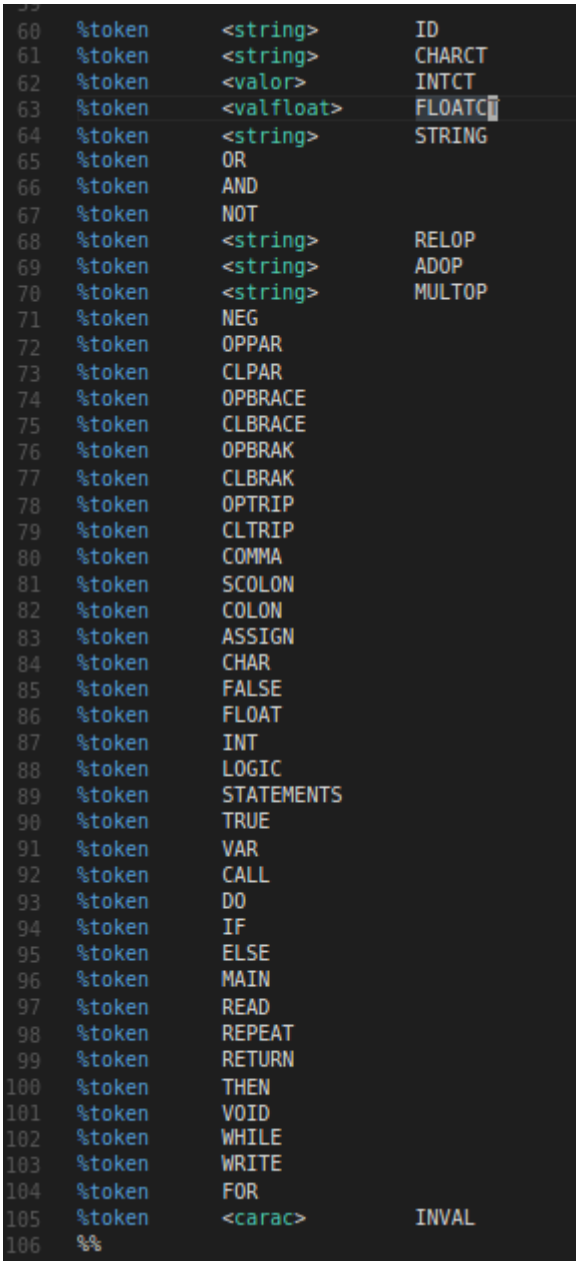
Instituto Tecnológico de Aeronáutica

I. Questão 1

Código:

```
/* Inclusao de arquivos da biblioteca de C */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Definicao dos atributos dos atomos operadores */

#define     LT          1
#define     LE          2
#define     GT          3
#define     GE          4
#define     EQ          5
#define     NE          6
#define     MAIS        7
#define     MENOS       8
#define     MULT        9
#define     DIV         10
#define     RESTO       11

/*   Definicao dos tipos de identificadores   */

#define     IDPROG      1
#define     IDVAR       2

/*  Definicao dos tipos de variaveis   */

#define     NAOVAR      0
#define     INTEIRO     1
#define     LOGICO      2
#define     REAL        3
#define     CARACTERE   4

/*   Definicao de outras constantes   */

#define NCLASSHASH  23
#define VERDADE     1
#define FALSO       0

int tab = 0;
int fromElse = 0;
int fromIf = 0;
void tabular(void);

%}
```

```
/* Definicao do tipo de yylval e dos atributos dos nao terminais */

%union {
    char string[50];
    int atr, valor;
    float valfloat;
    char carac;
}
```



```
%token      <string>        ID
%token      <string>        CHARCT
%token      <valor>         INTCT
%token      <valfloat>      FLOATCT
%token      <string>        STRING
%token      OR
%token      AND
%token      NOT
%token      <string>        RELOP
%token      <string>        ADOP
%token      <string>        MULTOP
%token      NEG
%token      OPPAR
%token      CLPAR
%token      OPBRACE
%token      CLBRACE
%token      OPBRAK
%token      CLBRAK
%token      OPTRIP
%token      CLTRIP
%token      COMMA
%token      SCOLON
%token      COLON
%token      ASSIGN
%token      CHAR
%token      FALSE
%token      FLOAT
%token      INT
%token      LOGIC
%token      STATEMENTS
%token      TRUE
%token      VAR
%token      CALL
%token      DO
%token      IF
%token      ELSE
%token      MAIN
%token      READ
%token      REPEAT
%token      RETURN
%token      THEN
%token      VOID
%token      WHILE
%token      WRITE
%token      FOR
%token      <carac>         INVAL
%%
```

```
Prog            :   ID  OPTRIP  {printf ("%s {{{\n", $1);}
                    Decls  ModList MainMod CLTRIP  {printf ("}}}\n");}
                ;
Decls           :
                |   VAR  OPBRACE  {tabular();tab++;printf ("var {\n");}  DeclList
                    CLBRACE  {tab--;tabular();printf ("}\n");}
                ;
DeclList        :   Declaration  |  DeclList  Declaration
                ;
Declaration     :   {tabular();} Type  ElemList  SCOLON  {printf (";\n");}
                ;
Type            :   INT  {printf ("int ");}
                |   FLOAT  {printf ("float ");}
                |   CHAR  {printf ("char ");}
                |   LOGIC  {printf ("logic ");}
                |   VOID {printf("void ");}
                ;
ElemList        :   Elem  |  ElemList  COMMA  {printf (", ");}  Elem
                ;
Elem            :   ID  {printf ("%s ", $1);} Dims
                ;
Dims            :
                |   OPBRAK {printf("[");} DimList CLBRAK {printf("]");}
                ;
DimList         :   INTCT  {printf ("%d", $1);}
                |   INTCT COMMA INTCT {printf("%d, %d", $1, $3);}
                |   INTCT COMMA INTCT COMMA INTCT {printf("%d, %d, %d", $1, $3, $5);}
                ;
ModList         :
                |   ModList Module
                ;
Module          :   ModHeader ModBody
                ;
ModHeader       :   Type ID OPPAR {printf("(");} CLPAR {printf(")");}
                |   Type ID OPPAR {printf("(");} ParamList CLPAR {printf(")");}
                ;
ParamList       :   Parameter
                |   ParamList COMMA {printf(", ");} Parameter
                ;
Parameter       :   Type ID
                ;
ModBody         :   Decls Stats
                ;
MainMod         :   MAIN ModBody
                ;
Stats           :   STATEMENTS  {tabular();printf ("statements ");}  CompStat
                ;
CompStat        :   OPBRACE  {printf ("{\n");tab++;} StatList  CLBRACE
                    {tab--;tabular();printf ("}\n");}
                ;
StatList        :
                |   StatList  Statement
                ;
```

```
Statement       :   CompStat  |  IfStat | WhileStat | RepeatStat | ForStat | ReadStat | WriteStat | AssignStat | CallStat | ReturnStat | SCOLON {tabular();printf(";");}
                ;
IfStat          :   IF {if(!fromElse)tabular();printf("if ");fromIf = 1;} Expression THEN {printf("then ");} Statement {fromIf = 0;}  ElseStat
                ;
ElseStat        :
                |   ELSE {tabular();printf("else ");fromElse = 1;} Statement {fromElse = 0;}
                ;
WhileStat       :   WHILE {tabular();printf("while ");} Expression DO {printf("do ");} Statement
                ;
RepeatStat      :   REPEAT {tabular();printf("repeat ");} Statement WHILE {tabular();printf("while ");} Expression SCOLON {printf(";\n");}
                ;
ForStat         :   FOR {tabular();printf("for ");} Variable OPPAR {printf("(");} AuxExpr4 COLON {printf(":");} Expression COLON {printf(":");} AuxExpr4 CLPAR {printf(")");
                ;
ReadStat        :   READ OPPAR {tabular();printf("read(");} ReadList CLPAR SCOLON {printf(");\n");}
                ;
ReadList        :   Variable
                |   ReadList COMMA {printf(", ");} Variable
                ;
WriteStat       :   WRITE OPPAR {tabular();printf("write(");} WriteList CLPAR SCOLON {printf(");\n");}
                ;
WriteList       :   WriteElem
                |   WriteList COMMA {printf(", ");} WriteElem
                ;
WriteElem       :   STRING {printf("%s", $1);}
                |   Expression
                ;
CallStat        :   CALL ID OPPAR {tabular();printf("call %s(", $2);} Arguments CLPAR SCOLON {printf(");\n");}
                ;
Arguments       :
                |   ExprList
                ;
ReturnStat      :   RETURN SCOLON {tabular();printf("return;\n");}
                |   RETURN {tabular();printf("return ");} Expression SCOLON {printf(";\n");}
                ;
AssignStat      :   {if(!fromElse)tabular();} Variable  ASSIGN  {printf (":= ");}  Expression  SCOLON
                    {printf (";\n");}
                ;
ExprList        :   Expression
                |   ExprList COMMA {printf(", ");} Expression
                ;
Expression      :   AuxExpr1  |  Expression  OR  {printf ("|| ");}  AuxExpr1
                ;
AuxExpr1        :   AuxExpr2  |  AuxExpr1  AND  {printf ("&& ");}  AuxExpr2
                ;
AuxExpr2        :   AuxExpr3  |  NOT  {printf ("! ");}  AuxExpr3
                ;
```

```
AuxExpr3        :   AuxExpr4
                |   AuxExpr4  RELOP  {
                        /*switch ($2) {
                            case LT: printf ("< "); break;
                            case LE: printf ("<= "); break;
                            case EQ: printf ("= "); break;
                            case NE: printf ("!= "); break;
                            case GT: printf ("> "); break;
                            case GE: printf (">= "); break;
                        }*/
                        printf("%s ", $2);
                    }  AuxExpr4
                ;
AuxExpr4        :   Term
                |   AuxExpr4  ADOP  {
                        /*switch ($2) {
                            case MAIS: printf ("+ "); break;
                            case MENOS: printf ("- "); break;
                        }*/
                        printf("%s ", $2);
                    }  Term
                ;
Term            :   Factor
                |   Term  MULTOP  {
                        /*switch ($2) {
                            case MULT: printf ("* "); break;
                            case DIV: printf ("/ "); break;
                            case RESTO: printf ("%% "); break;
                        }*/
                        printf("%s ", $2);
                    }  Factor
                ;
Factor          :   Variable
                |   INTCT  {printf ("%d ", $1);}
                |   FLOATCT  {printf ("%g ", $1);}
                |   CHARCT  {printf ("%s ", $1);}
                |   TRUE  {printf ("true ");}
                |   FALSE  {printf ("false ");}
                |   NEG  {printf ("~ ");}  Factor
                |   OPPAR  {printf ("( ");}  Expression  CLPAR  {printf (") ");}
                |   FuncCall
                ;
Variable        :   ID  {printf ("%s ", $1);} Subscripts
                ;
Subscripts      :
                |   OPBRAK {printf("[");} SubscrList CLBRAK {printf("]");}
                ;
SubscrList      :   AuxExpr4
                |   TwoSubscr
                |   ThreeSubscr
                ;
```

```
TwoSubscr       :   AuxExpr4 COMMA {printf(", ");} AuxExpr4
                ;
ThreeSubscr     :   TwoSubscr COMMA {printf(", ");} AuxExpr4
                ;
FuncCall        :   ID OPPAR {printf("%s(",$1);} Arguments CLPAR {printf(")");}
                ;
%%

/* Inclusao do analisador lexico  */

#include "lex.yy.c"

void tabular() {
    int i;
    for(i = 1; i <= tab; i++){
        printf("\t");
    }
}
```

Análise:

Foi usada a entrada existente no arquivo com as especificações da linguagem COMP-2018. Além disso, foi usado o mesmo arquivo lexx do laboratório 2. O programa foi executado e gerou o arquivo de saída printado corretamente que se encontra no github do autor, na pasta do laboratório.

Nessa pasta, se encontram os arquivos lex e yacc, o arquivo de entrada, o arquivo de saída e o arquivo do relatório.