Processamento de Linguagens
Relatório do Segundo Exercício Prático

Maria Ana de Brito (73580), Bruno Pereira (75135), Diogo Silva (76407)

Universidade do Minho

Compilador de uma Linguagem Imperativa Simples

Abstract. Este relatório detalha a especificação de todo o trabalho efetuado pelo grupo na resolução de um dos exercícios propostos. Assim, justificar-se-ão todas as decisões tomadas, bem como todo o código desenvolvido. Por fim, proceder-se-á às conclusões finais do trabalho realizado.

Table of Contents

1 Introdução

Este trabalho prático tem como objetivo aprofundar os conhecimentos sobre *Yacc*, bem como o desenho de gramáticas e analisadores léxicos, conhecimentos estes obtidos ao longo das aulas práticas. Deste modo, utilizou-se a linguagem C, o *Flex* e o *Yacc*, visando a criação de um compilador que utiliza uma gramática independente de contexto, capaz de reconhecer uma linguagem de programação imperativa simples. Ao longo deste relatório, está desenvolvido o método de como foi implementada esta GIC, bem como as estruturas de dados utilizadas, acabando com o conjunto dos testes pedidos no enunciado do trabalho.

2 Desenho da gramática

A primeira ação a ser realizada foi a construção da GIC(Gramática Independente de Contexto). Assim, para a definição da linguagem imperativa requerida foi necessário definir os símbolos terminais, os símbolos não terminais, o axioma da gramática e as regras de produção. A frase anterior pode ser descrita pela seguinte fórmula G=<T,N,S,P>.

Relativamente aos símbolos terminais(T), é importante referir o *num* (cuja expressão regular é definida da seguinte forma: [0-9]+) e representa um número inteiro, a *var* (cuja expressão regular é definida da seguinte forma: [a-z]+) e representa uma variável da linguagem imperativa e a *string* (cuja expressão regular é definida da seguinte forma: "[^"]+") e que representa uma *string* de texto. O conjunto total de símbolos terminais é:

T={ 'DECLARACOES', 'GET', 'GO', 'STOP', 'PUT', 'WHILE', 'IF', 'ELSE', 'ERRO', 'MULT', 'MULT', 'ADD', 'SUB', 'DIV', 'REST', 'GREATER', 'LESSER', 'EQUAL', 'DIFFERENT', 'LTE', 'GTE', 'AND', 'OR', 'NOT', var, num, STRING, '[', ']', ';', '{', '}', '=', '(', ')', }.

Relativamente aos símbolos não terminais(N), temos que o conjunto de todos os símbolos terminais é:

N={ Prog, Cabec, Corpo, Decs, Dec, Insts, Inst, Atrib, Escrita, Condicao, Ciclo, Leitura, Exps, Exp, If, Else }.

O axioma da linguagem (S) é definido por S = { Prog }.

O conjunto das produções é representado pela letra P e definido pelo seguinte conjunto:
P={

Prog → Cabec Corpo
Cabec → ϵ
Cabec → DECLARACOES Decs
Decs → Dec
Decs → DecDecs Dec

Dec → var ';'
Dec → VAR '[' num ']' ';'
Corpo → ϵ
Corpo → GO Insts STOP
Insts → Inst
Insts → Insts Inst
Inst → Atribuicao
Inst → Escrita
Inst → Condicao
Inst → Condicional
Inst → Leitura
Leitura → GET var ';'
Leitura → GET var'[' Exp ']' ';'
Atrib → var '=' Exps ';'
Atrib → var '[' Exp ']' '=' Exps ';'
Escrita → PUT Exps ';'
Escrita → PUT STRING ';'
Condicao → IF Else
If → IF '(' ExpBool ')' '{'Insts '}'
Else → ELSE '{'Insts '}'
Ciclo → WHILE '(' ExpBool ')' '{' Insts '}'
ExpBool → Exps GREATER Exps
ExpBool → Exps LESSER Exps
ExpBool → Exps EQUAL Exps
ExpBool → Exps DIFFERENT Exps
ExpBool → Exps LTE Exps
ExpBool → Exps GTE Exps
ExpBool → NOT Exp
Exps → Exp
Exps → Exps MULT Exp
Exps → Exps ADD Exp
Exps → Exps SUB Exp
Exps → Exps DIV Exp
Exps → Exps REST Exp
Exps → '(' Exps ')'
Exp → var
Exp → var '[' Exp ']'
Exp → num

}

3 Desenho da solução

3.1 Estruturas de dados

As estruturas de dados utilizadas têm como objetivo armazenar a informação crucial para a correta geração de pseudo-código *Assembly*, existindo estruturas para armazenar informação relativa a variáveis, instruções, saltos condicionais e ciclos. Todas as estruturas são listas ligadas, sendo que as estruturas referentes aos saltos condicionais e aos ciclos, funcionam como *stacks*.

Variáveis A estrutura que contém informação sobre as variáveis apresenta o endereço das variáveis, o nome e um inteiro (quase um *booleano*) que indica se a variável foi inicializada (controlo de erros). Por fim, apresenta um endereço para a próxima variável.

```
typedef struct variavel
{
int endr;
char* nome;
int init;
struct variavel *prox;
}Variavel;
```

Instruções A estrutura que contém informação sobre as instruções apresenta o endereço da instrução, o nome da instrução e um endereço para a próxima instrução.

```
typedef struct instrucao
{
int endr;
char* nome;
struct instrucao *prox;
}Instrucao;
```

If e Else A estrutura que armazena a informação sobre os saltos condicionais apresenta o endereço do salto, caso a expressão falhe e uma etiqueta, que indica que tipo de salto é. Finalmente é apresentado um endereço para (a próxima estrutura do) próximo salto.

```
typedef struct ifelse
{
int endr;
char* etiqueta;
struct ifelse *prox;
}IfElse;
```

Ciclo *While* A estrutura que armazena a informação sobre os ciclos apresenta o endereço o início do ciclo, o endereço da instrução a seguir ao ciclo, ou seja, o endereço para sair do ciclo e o endereço para a próxima estrutura representativa do ciclo.

```
typedef struct whilecicle
{
int endr_inicio;
int endr_fim;
struct whilecicle *prox;
}WhileCicle;
```

3.2 Funções definidas em C

Apesar da maioria do código ter sido efetuado dentro do ficheiro *Yacc* existem alguma funções que permitem efetuar um controlo sobre as estruturas de dados, sendo elas:

- **addVar*: adiciona uma variável à estrutura de variáveis;
- *endVar*: retorna o endereço de uma variável, ou, caso ela não exista, um valor de erro;
- *varInit*: declara uma variável como inicializada;
- *isVarInit*: verifica se uma variável está inicializada;
- **addInst*: adiciona uma instrução à estrutura de instruções.

article [utf8]inputenc

4 Testes e Resultados

4.1 Ler 4 números e dizer se podem ser os lados de um quadrado

Neste teste são lidos 4 números e determina-se se podem ser os lados de um quadrado, isto é, se têm o mesmo valor. Este é o código definido:

```
DECLARACOES
x;
y;
z;
w;
flag;
GO
GET x;
GET y;
GET z;
GET w;
flag = 0;
IF(x == y)
{
    IF(x == z)
    {
        IF(x == w)
        {
            flag = 1;
        }
    }
}
IF(flag == 0)
{
     PUT "Os números não representam os lados de um quadrado!";
```

```
}
ELSE
{
     PUT "Os números representam os lados de um quadrado!";
}

STOP
```

E este é o código assembly gerado:

```
0: pushi 0
i1: pushi 0
i2: pushi 0
i3: pushi 0
i4: pushi 0
i5: start
i6: read
i7: atoi
i8: storeg 0
i9: read
i10: atoi
i11: storeg 1
i12: read
i13: atoi
i14: storeg 2
i15: read
i16: atoi
i17: storeg 3
i18: pushi 0
i19: storeg 4
i20: pushg 0
i21: pushg 1
i22: equal
i23: jz i35
i24: pushg 0
i25: pushg 2
i26: equal
i27: jz i35
i28: pushg 0
i29: pushg 3
i30: equal
i31: jz i35
i32: pushi 1
i33: storeg 4
```

```
i34: pushg 4
i35: pushi 0
i36: equal
i37: jz i41
i38: pushs "Os números não representam os lados de um quadrado!"
i39: writes
i40: jump i43
i41: pushs "Os números representam os lados de um quadrado!"
i42: writes
i43: stop
```

Com os números de input 3, 5, 6 e 4, temos que estes não podem representar um quadrado. Por outro lado, com os números 4, 4, 4 e 4, temos que estes podem representar os lados de um quadrado.

4.2 Ler um inteiro N, depois ler N números e escrever o menor deles

Neste teste é lido primeiro um número inteiro N e depois são lidos N números e determina-se o menor deles. Este é o código definido:

```
DECLARACOES
n;
i;
x;
min;
GO
GET n;
GET min;
i=1;
WHILE(i < n)
{
    GET x;
    IF(min > x)
    {
         min = x;
    }
    ELSE
    {
         x = 0;
    }

    i = i+1;
}
PUT "O menor número é o ";
PUT min;
```

```
STOP
```

E este é o código assembly gerado:

```
i0: pushi 0
i1: pushi 0
i2: pushi 0
i3: pushi 0
i4: start
i5: read
i6: atoi
i7: storeg 0
i8: read
i9: atoi
i10: storeg 3
i11: pushi 1
i12: storeg 1
i13: pushg 1
i14: pushg 0
i15: inf
i16: jz i34
i17: read
i18: atoi
i19: storeg 2
i20: pushg 3
i21: pushg 2
i22: sup
i23: jz i27
i24: pushg 2
i25: storeg 3
i26: jump i29
i27: pushi 0
i28: storeg 2
i29: pushg 1
i30: pushi 1
i31: add
i32: storeg 1
i33: jump i13
i34: pushs "O menor número é o "
i35: writes
i36: pushg 3
i37: writei
i38: stop
```

Se o número N lido for 4 e os restantes número forem 5, 2, 7, e 6, é-nos indicado que o menor número desta sequência é o 2.

4.3 Ler N números e calcular e imprimir o seu produtório

Neste teste lemos um número pré-definido de números e calcula-se o seu produtório. Este é o código definido:

```
DECLARACOES
n;
prod;
i;
x;
GO
n = 4;
i = 0;
prod=1;
while(i < n)
{
    GET x;
    prod = prod*x;
    i = i+1;
}
PUT "O produtório dos números lidos é ";
PUT prod;
STOP
```

E este é o assembly gerado:

```
i0: pushi 0
i1: pushi 0
i2: pushi 0
i3: pushi 0
i4: start
i5: pushi 4
i6: storeg 0
i7: pushi 0
i8: storeg 2
i9: pushi 1
i10: storeg 1
i11: pushg 2
i12: pushg 0
i13: inf
i14: jz i27
i15: read
i16: atoi
i17: storeg 3
i18: pushg 1
i19: pushg 3
i20: mul
```

```
i21: storeg 1
i22: pushg 2
i23: pushi 1
i24: add
i25: storeg 2
i26: jump i11
i27: pushs "O produtório dos números lidos é "
i28: writes
i29: pushg 1
i30: writei
i31: stop
```

Sendo 4 o número pré-definido de números a serem lidos, temos que o produtório de 1, 2, 3 e 4 é 24.

4.4 Contar e imprimir os números ímpares de uma sequência de números naturais

Neste teste inserimos um número N e lemos N números de input e são nos indicados quais e quantos são ímpares. Este é o código definido:

```
DECLARACOES
n;
x;
y;
acc;
i;
GO
GET n;
PUT "Os números ímpares são:\n";
i=0;
while(i < n)
{
    GET x;
    if(x%2 > 0)
    {
        PUT x;
        PUT "  \n";
        acc = acc+1;
    }
    else
    {
        x = 0;
    }
    i = i+1;
}
```

```
PUT "\nO total de números ímpares é: ";
PUT acc;
STOP
```

E este é o código assembly gerado:

```
i0: pushi 0
i1: pushi 0
i2: pushi 0
i3: pushi 0
i4: pushi 0
i5: start
i6: read
i7: atoi
i8: storeg 0
i9: pushs "Os números ímpares são:\n"
i10: writes
i11: pushi 0
i12: storeg 4
i13: pushg 4
i14: pushg 0
i15: inf
i16: jz i42
i17: read
i18: atoi
i19: storeg 1
i20: pushg 1
i21: pushi 2
i22: mod
i23: pushi 0
i24: sup
i25: jz i35
i26: pushg 1
i27: writei
i28: pushs "  \n"
i29: writes
i30: pushg 3
i31: pushi 1
i32: add
i33: storeg 3
i34: jump i37
i35: pushi 0
i36: storeg 1
i37: pushg 4
i38: pushi 1
```

```
i39: add
i40: storeg 4
i41: jump i13
i42: pushs "\nO total de números ímpares é: "
i43: writes
i44: pushg 3
i45: writei
i46: stop
```

Com a sequência 4, 5, 8, 7 e 1, os números ímpares são 5, 7 e 1.

4.5 Ler e armazenar N números num array e imprimir os valores por ordem inversa

Neste teste inserimos um número pré-definido de números de input, sendo, então, mostrado a mesma sequência na ordem inversa. Este é o código definido:

```
DECLARACOES
array[5];
x;
i;
j;
GO
i = 0;
WHILE(i < 5)
{
    GET array[i];
    i = i+1;
}
i = 4;
PUT "Os valores por ordem inversa são:\n";
PUT "[";

WHILE(i > 0)
{
    PUT array[i];
    if(i > 1)
    {
        PUT ", ";
    }
    i = i-1;
}
PUT ", ";
PUT array[0];
PUT "]";
STOP
```

E este é o código assembly gerado:

```
i0: pushn 5
i1: pushi 0
i2: pushi 0
i3: pushi 0
i4: start
i5: pushi 0
i6: storeg 6
i7: pushg 6
i8: pushi 5
i9: inf
i10: jz i23
i11: pushgp
i12: pushi 0
i13: padd
i14: pushg 6
i15: read
i16: atoi
i17: storen
i18: pushg 6
i19: pushi 1
i20: add
i21: storeg 6
i22: jump i7
i23: pushi 4
i24: storeg 6
i25: pushs "Os valores por ordem inversa são:\n"
i26: writes
i27: pushs "["
i28: writes
i29: pushg 6
i30: pushi 0
i31: sup
i32: jz i50
i33: pushgp
i34: pushi 0
i35: padd
i36: pushg 6
i37: loadn
i38: writei
i39: pushg 6
i40: pushi 1
i41: sup
```

```
i42: jz i46
i43: pushs ", "
i44: writes
i45: pushg 6
i46: pushi 1
i47: sub
i48: storeg 6
i49: jump i29
i50: pushs ", "
i51: writes
i52: pushgp
i53: pushi 0
i54: padd
i55: pushi 0
i56: loadn
i57: writei
i58: pushs "]"
i59: writes
i60: stop
```

Neste teste, temos que quando a sequência 3, 4, 5, 6 e 7, a sequência inversa é 7, 6, 5, 4 e 3.

4.6 Conclusão

Concluindo o trabalho prático, o grupo avalia o seu desempenho como positivo. Apesar de existirem alguns aspetos a serem melhorados, o grupo conseguiu realizar a tarefa proposta pela equipa docente. Assim sendo, o grupo criou uma gramática, bem como o seu analisador léxico. Além disso, foram geradas instruções assembly para testar na máquina virtual disponibilizada. As dificuldades deste trabalho surgiram maioritariamente nas instruções assembly, pois é um aspeto no qual o grupo não tem muita experiência. Deste modo, grande parte do tempo do desenvolvimento deste segundo trabalho prático foi gasto a compreender quais seriam as instruções adequadas e qual a sua ordem. Em suma, o grupo sente que conseguiu consolidar os conhecimentos adquiridos ao longo das aulas e adquiriu mais facilidade ao trabalhar com instruções assembly.

A Código Yacc

```
%{
#include <stdio.h>
#include <string.h>
#include "variavel.c"
#include "instrucao.c"
#include "while.h"
#include "ifelse.h"

extern int yylineno;

int yylex();
int yyerror(char *s);

Variavel *variaveis = NULL;
Variavel *auxVar = NULL;
Instrucao *instrucoes = NULL;
Instrucao *auxInst = NULL;
WhileCicle *whiles = NULL;
WhileCicle *auxWhile = NULL;
IfElse *ifelses = NULL;
IfElse *auxIfElse = NULL;

int globalpointer;
int nInst = 0;
int nWhile = 0;
int erro = 0;

%}

%union{int numero; char *string; char *variavel;}

%token <numero> num
%token <variavel>var
%token <string>STRING
%token MULT ADD SUB DIV REST GREATER LESSER EQUAL DIFFERENT LTE GTE AND OR NOT
%token DECLARACOES GET GO STOP PUT WHILE IF ELSE ERRO

%start Prog

%%

Prog : Cabec Corpo
 ;
```

```
Cabec :
  | DECLARACOES Decs {
   auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("start");

   instrucoes = addInst(auxInst, instrucoes);
     }
  ;

Decs : Dec
 | Decs Dec
 ;

Dec : var ';' { int endr;
endr = endrVar($1, variaveis);
if(endr == -1) //Se não existir
{
auxVar = malloc(sizeof(struct variavel));
auxVar->endr = globalpointer++;
auxVar->nome = strdup($1);
auxVar->init = 0;

variaveis = addVar(auxVar, variaveis);

auxInst = malloc(sizeof(struct instrucao));
auxInst->endr = nInst++;
auxInst->nome = strdup("pushi 0");

instrucoes = addInst(auxInst, instrucoes);
}
else
{
erro = 1;
fprintf(stdout, "Linha %d: Variável %s já está declarada!\n", yylineno, $1);
}
  }
| var '[' num ']' ';' {
int endr;
endr = endrVar($1, variaveis);

if(endr == -1) //Se não existir
{
auxVar = malloc(sizeof(struct variavel));
auxVar->endr = globalpointer;
```

```
auxVar->nome = strdup($1);
auxVar->init = 0;

variaveis = addVar(auxVar, variaveis);

char buffer[50];
sprintf(buffer, "pushn %d", $3);

auxInst = malloc(sizeof(struct instrucao));
auxInst->endr = nInst++;
auxInst->nome = strdup(buffer);

instrucoes = addInst(auxInst, instrucoes);

globalpointer += $3;
}
else
{
erro = 1;
fprintf(stdout, "Linha %d: Variável %s já está declarada!\n", yylineno, $1);
}
  }
;

Corpo :
  | GO Insts STOP {
   auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("stop");

   instrucoes = addInst(auxInst, instrucoes);
     }
  ;

Insts : Inst
  | Insts Inst
  ;

Inst : Atrib
 | Escrita
 | Condicao
 | Ciclo
 | Leitura
 ;
```

```
Leitura : GET var {
int endr;
endr = endrVar($2, variaveis);

if(endr == -1) //Se não existir
{
erro = 1;
fprintf(stdout, "Linha %d: Variável %s não está declarada!\n", yylineno, $2);
}
else
{
varInit($2, variaveis); // Se existir, marca-se como inicializada

auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("read");

   instrucoes = addInst(auxInst, instrucoes);

   auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("atoi");

   instrucoes = addInst(auxInst, instrucoes);

   char buffer[50];
   sprintf(buffer, "storeg %d", endr);

   auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

   instrucoes = addInst(auxInst, instrucoes);
}
  } ';'
        | GET var {
         int endr;
endr = endrVar($2, variaveis);

if(endr == -1) //Se não existir
{
erro = 1;
fprintf(stdout, "Linha %d: Variável %s não está declarada!\n", yylineno, $2);
}
else
```

```
{
varInit($2, variaveis); // Se existir, marca-se como inicializada

char buffer[50];

sprintf(buffer, "pushgp");

auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

instrucoes = addInst(auxInst, instrucoes);

sprintf(buffer, "pushi %d", endr);

auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

instrucoes = addInst(auxInst, instrucoes);

sprintf(buffer, "padd");

auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

instrucoes = addInst(auxInst, instrucoes);

}
           }'[' Exp ']' ';' {

            auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("read");

   instrucoes = addInst(auxInst, instrucoes);

   auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("atoi");

   instrucoes = addInst(auxInst, instrucoes);

   auxInst = malloc(sizeof(struct instrucao));
```

```
   auxInst->endr = nInst++;
   auxInst->nome = strdup("storen");

   instrucoes = addInst(auxInst, instrucoes);
            }
        ;

Atrib : var '=' Exps ';' {
int endr;
endr = endrVar($1, variaveis);

if(endr == -1) //Se não existir
{
erro = 1;
fprintf(stdout, "Linha %d: Variável %s não está declarada!\n", yylineno, $1);
}
else
{
varInit($1, variaveis); // Se existir, marca-se como inicializada

char buffer[50];
sprintf(buffer, "storeg %d", endr);

auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

instrucoes = addInst(auxInst, instrucoes);
}
 }
  | var {
int endr;
endr = endrVar($1, variaveis);

if(endr == -1) //Se não existir
{
erro = 1;
fprintf(stdout, "Linha %d: Variável %s não está declarada!\n", yylineno, $1);
}
else
{
varInit($1, variaveis); // Se existir, marca-se como inicializada

char buffer[10];
sprintf(buffer, "pushi %d", endr);
```

```
auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

instrucoes = addInst(auxInst, instrucoes);
}
   }
   '[' Exp ']' '=' Exps ';' {
   auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = "storen";

   instrucoes = addInst(auxInst, instrucoes);
     }
  ;

Escrita : PUT Exps ';' {
      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("writei");

      instrucoes = addInst(auxInst, instrucoes);
   }
| PUT STRING ';' {
char buffer[1024];
sprintf(buffer, "pushs \"%s\"", $2);

      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

      instrucoes = addInst(auxInst, instrucoes);
      auxInst = malloc(sizeof(struct instrucao));
      auxInst->endr = nInst++;
      auxInst->nome = strdup("writes");

      instrucoes = addInst(auxInst, instrucoes);
          }
;

Condicao : If Else;

If : IF '(' ExpBool ')' {
auxIfElse = malloc(sizeof(struct ifelse));
```

```
auxIfElse->endr = nInst;
auxIfElse->prox = ifelses;
ifelses = auxIfElse;

auxInst = malloc(sizeof(struct ifelse));
auxInst->endr = nInst++;
auxInst->nome = strdup("");
auxInst->prox = instrucoes;
instrucoes = auxInst;
}
'{' Insts '}' {
auxInst = instrucoes;
int flag = 0;
while(auxInst && flag == 0)
{
if(auxInst->endr == ifelses->endr)
{
flag = 1;
}
else
auxInst = auxInst->prox;
}

char buffer[50];
sprintf(buffer, "jz i%d", nInst+1);
auxInst->nome = strdup(buffer);

auxIfElse = ifelses->prox;
free(ifelses);
ifelses = auxIfElse;
  }
   ;

Else :
 | ELSE {
  auxIfElse = malloc(sizeof(struct ifelse));
auxIfElse->endr = nInst;
auxIfElse->prox = ifelses;
ifelses = auxIfElse;

auxInst = malloc(sizeof(struct ifelse));
auxInst->endr = nInst++;
auxInst->nome = strdup("");
auxInst->prox = instrucoes;
instrucoes = auxInst;
```

```
  }
 '{' Insts '}' {
  auxInst = instrucoes;
int flag = 0;
while(auxInst && flag == 0)
{
if(auxInst->endr == ifelses->endr)
{
flag = 1;
}
else
auxInst = auxInst->prox;
}

char buffer[50];
sprintf(buffer, "jump i%d", nInst);
auxInst->nome = strdup(buffer);

auxIfElse = ifelses->prox;
free(ifelses);
ifelses = auxIfElse;
     }
 ;

Ciclo : WHILE {
nWhile = nInst;
  }
'(' ExpBool {
auxWhile = malloc(sizeof(struct whilecicle));
auxWhile->endr_inicio = nWhile;
auxWhile->endr_fim = nInst;
auxWhile->prox = whiles;
whiles = auxWhile;

auxInst = malloc(sizeof(struct instrucao));
auxInst->endr = nInst++;
auxInst->nome = strdup("");

instrucoes = addInst(auxInst, instrucoes);
}
')' '{' Insts {
auxInst = instrucoes;
int flag = 0, endr_i;
while(auxInst && flag == 0)
{
```

```
if(auxInst->endr == whiles->endr_fim)
{
flag = 1;
endr_i = whiles->endr_inicio;
}
else
auxInst = auxInst->prox;
}

char buffer[50];
sprintf(buffer, "jz i%d", nInst+1); // Endereço depois do ciclo while
auxInst->nome = strdup(buffer);

auxWhile = whiles->prox;
free(whiles);
whiles = auxWhile;

sprintf(buffer, "jump i%d", endr_i);
auxInst = malloc(sizeof(struct instrucao));
auxInst->endr = nInst++;
auxInst->nome = strdup(buffer);
instrucoes = addInst(auxInst, instrucoes);

  }
'}'
  ;

ExpBool : Exps GREATER Exps {
      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("sup");

      instrucoes = addInst(auxInst, instrucoes);
  }
| Exps LESSER Exps {
      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("inf");

      instrucoes = addInst(auxInst, instrucoes);
         }
| Exps EQUAL Exps {
auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("equal");
```

```
    instrucoes = addInst(auxInst, instrucoes);
  }
| Exps DIFFERENT Exps {
auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("equal");

      instrucoes = addInst(auxInst, instrucoes);

      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("not");

      instrucoes = addInst(auxInst, instrucoes);
  }
| Exps LTE Exps {
auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("infeq");

      instrucoes = addInst(auxInst, instrucoes);
}
| Exps GTE Exps {
auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("supeq");

    instrucoes = addInst(auxInst, instrucoes);
}
| NOT Exp {
      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("not");

      instrucoes = addInst(auxInst, instrucoes);
            }
;

Exps : Exp
 | Exps MULT Exp {
      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("mul");
```

```
      instrucoes = addInst(auxInst, instrucoes);
           }
 | Exps ADD Exp {
      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("add");

      instrucoes = addInst(auxInst, instrucoes);
          }
 | Exps SUB Exp {
      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("sub");

      instrucoes = addInst(auxInst, instrucoes);
          }
 | Exps DIV Exp {
      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("div");

      instrucoes = addInst(auxInst, instrucoes);
          }
 | Exps REST Exp {
      auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("mod");

      instrucoes = addInst(auxInst, instrucoes);
           }
 | '(' Exps ')'
 ;

Exp : var {
int endr;
endr = endrVar($1, variaveis);

if(endr == -1) //Se não existir
{
erro = 1;
fprintf(stdout, "Linha %d: Variável %s não está declarada!\n", yylineno, $1);
}
else
{
varInit($1, variaveis); // Se existir, marca-se como inicializada
```

```
char buffer[50];
sprintf(buffer, "pushg %d", endr);

auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

instrucoes = addInst(auxInst, instrucoes);
}
  }
| var {
int endr;
endr = endrVar($1, variaveis);

if(endr == -1) //Se não existir
{
erro = 1;
fprintf(stdout, "Linha %d: Variável %s não está declarada!\n", yylineno, $1);
}
else
{
varInit($1, variaveis); // Se existir, marca-se como inicializada

char buffer[50];

sprintf(buffer, "pushgp");
auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

instrucoes = addInst(auxInst, instrucoes);

sprintf(buffer, "pushi %d", endr);
auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

instrucoes = addInst(auxInst, instrucoes);

sprintf(buffer, "padd");
auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);
```

```
instrucoes = addInst(auxInst, instrucoes);
}
  }
 '[' Exp ']' {
  auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup("loadn");

  instrucoes = addInst(auxInst, instrucoes);
   }
| num {
char buffer[50];
sprintf(buffer, "pushi %d", $1);

auxInst = malloc(sizeof(struct instrucao));
   auxInst->endr = nInst++;
   auxInst->nome = strdup(buffer);

instrucoes = addInst(auxInst, instrucoes);
 }
;


%%

int yyerror(char *s)
{
printf("erro:%s\n", s);
return 0;
}

int main(int argc, char* argv[])
{

yyparse();
int i, count = 0;
auxInst = instrucoes;
while(auxInst)
{
count++;
auxInst = auxInst->prox;
}

char** insts;
insts = malloc(count*sizeof(char*));
```

```
for(i = 0; i < count; i++, instrucoes= instrucoes->prox)
{
char buffer[1024];
sprintf(buffer, "i%d: %s\n", instrucoes->endr, instrucoes->nome);

insts[count-1- i] = strdup(buffer);
}


if(erro == 0)
{
for(i = 0; i < count; i++)
{
printf("%s", insts[i]);
}
}
else printf("Frase incorreta!\n");



return 0;
}
```

B Analisador Léxico

```
%{
#include "y.tab.h"
%}

%option noyywrap
%option yylineno

string  \"([^"]|\\\")*\"

%%
[ \t\n] {;}
[\(\)\[\]\{\}\;\"\=] {return yytext[0];}
(?i:DECLARACOES) {return DECLARACOES;}
(?i:GO) {return GO;}
(?i:STOP) {return STOP;}
(?i:GET) {return GET;}
(?i:PUT) {return PUT;}
{string} { yylval.string = strdup(yytext+1);
yylval.string[strchr(yylval.string,'"')-yylval.string] = '\0';
return STRING;
```

```
}
(?i:IF) {return IF;}
(?i:ELSE) {return ELSE;}
(?i:WHILE) {return WHILE;}
\! {return NOT;}
\> {return GREATER;}
\< {return LESSER;}
\=\= {return EQUAL;}
\!\= {return DIFFERENT;}
\<\= {return LTE;}
\>\= {return GTE;}
\&\& {return AND;}
\|\| {return OR;}
\* {return MULT;}
\+ {return ADD;}
\- {return SUB;}
\/ {return DIV;}
\% {return REST;}
[0-9]+ {yylval.numero = atoi(strdup(yytext));
return num;}
[a-z]+ {yylval.variavel = strdup(yytext);
return var;}
. {return ERRO;}
```