Processamento de Linguagens (3º ano de MiEI)

Trabalho Prático 2

Relatório de Desenvolvimento de uma *LPIS*

Gustavo da Costa Gomes-Aluno (72223)

José Carlos da Silva Brandão Gonçalves-Aluno (71223)

Tiago João Lopes Carvalhais-Aluno (70443)

May 30, 2016

Abstract

Isto é o resumo do relatório da unidade curricular Processamento de Linguagens relativamente ao Trabalho Prático 2.

Este visa a produção de uma linguagem de programação imperativa simples, *LPIS*, e para tal será necessário produzir uma gramática independente de contexto *GIC* com condição LR(0), bem como a produção do respetivo compilador, com auxílio do Gerador *Yacc/Flex* e da realização de um conjuntos de testes, escritos na linguagem especificada na *GIC*, por forma a verificar o seu correto funcionamento.

E esse compilador, no final, produzirá pseudo-código assembly, que será interpretado pela máquina virtual e que gerará o resultado esperado.

Contents

Chapter 1

Introdução

Neste trabalho todos os parâmetros exigidos foram realizados, bem como a realização de um conjunto de testes que foram para além dos seis testes que foram estabelecidos como requisitos minimos.

A realização desses parâmetros passou, inicialmente, por produzir uma gramática independente de contexto *GIC* e, após a sua validação pelo docente Pedro Rangel Henriques, por criar um compilador que reconhecesse um ficheiro de entrada, escrito nessa linguagem, e que produzisse o ficheiro .vm correspondente, recorrendo ao Gerador *Yacc/Flex*. Ficheiro esse que contém o conjunto das instruções assembly para a máquina virtual.

Enquadramento Utilização do Gerador *Yacc/Flex* e Desenvolvimento de uma *GIC* com o objetivo de produzir pseudo-código para a máquina virtual a partir dos padrões da linguagem produzida existentes num outro ficheiro.

Estrutura do documento Este documento possui a respectiva solução elaborada e uma conclusão final que une o exercício.

Resultados Os resultados serão apreciados no capítulo vindouro e cuja solução estará em Anexo.

Conteúdo do documento Contém a explicação do problema em si, bem como a apresentação da solução produzida para produzir o resultado esperado, auxiliada com documentação e código produzido.

1.1 Estrutura do Relatório

Este relatório possui três capítulos, uma conclusão,um capítulo extra dedicado a Anexos e a respectiva Bibliografia utilizada durante a realização deste projecto.

Os capítulos são Análise e Especificação do problema, Arquitectura da solução para o problema em questão, que envolverá a explicação das estruturas de dados utilizadas e por fim o capítulo designado Codificação, que incluirá alguns aspectos relevantes de todos os testes realizados para a verificação do seu correcto funcionamento.

Chapter 2

Análise e Especificação

2.1 Descrição informal do problema

Este trabalho prático pretende que o compilador produzido seja capaz de construir ficheiros com o pseudo-código assembly para a máquina de stack virtual(VM) fornecido pelos docentes.

Esse compilador para realizar a sua função em pleno necessita de receber como argumento ficheiros escritos com a linguagem que também terá de ser desenvolvida e validada por um dos docentes. A linguagem a produzir deverá permitir:

- Declaração e manuseamento de variáveis atómicas do tipo inteiro, com os quais se podem realizar as habituais operações aritméticas, relacionais e lógicas.
- Declaração e manuseamento de variáveis estruturadas do tipo array (a 1 ou 2 dimensões, matrizes) de inteiros, em relação aos quais é apenas permitida a operação de indexação (índice inteiro).
- Realização de instruções algorítmicas básicas como o caso da atribuição de expressões a variáveis.
- Leitura do standard input e escrita no standard output.
- Realização de instruções para controlo do fluxo de execução, condicional e cíclica, que possam ser aninhadas.
- Definição e invocação de sub programas sem parâmetros, mas que possam retornar um resultado atómico, sendo este tópico opcional. Contudo também será produzido, e testado, por exemplo, a função "potencia(Base,Exp)".

Esta linguagem deverá ainda ter em conta que as variáveis deverão ser declaradas no início do programa e que não se poderá efetuar declarações nem utilizações sem declaração prévia das mesmas. Se nada for dito em relação ao valor que a variável toma, essa toma, por defeito, o valor zero, após a sua declaração.

Após se produzir essa gramática, a mesma deverá ser validada antes de se prosseguir. E por fim, após a validação dessa é necessário realizar as análises léxica e sintática acompanhada pelas respectivas ações a tomar que resultarão na criação de um ficheiro com as instruções assembly correspondentes, ordenadas corretamente por forma a produzirem o resultado final devido pela máquina de stack virtual.

2.2 Especificação de Requisitos

Para iniciar a construção da gramática, em primeiro lugar, deve-se ter em conta a definição dos símbolos terminais e não terminais, adicionando as regras de produção necessárias.

Posto isto, seja G = <T, NT, S, P>a gramática para a linguagem imperativa simples (LPIS), onde T é o conjunto dos símbolos terminais, NT o conjunto dos símbolos não ter- minais, S o axioma da gramática e P o conjunto das regras de produção.

Seja NUMBER a expressão regular que representa um número inteiro, NAME a que representa um nome composto por carateres do alfabeto, podendo também conter o carater '_' e TEXT a que representa todo o tipo de carateres de texto. Para evitar conflitos, optámos por criar duas novas palavras reservadas, D1 e D2, que indicam a dimensão de um array. AND, OR, EQLS, DIFF, SMEQ e GTEQ representam os símbolos "&&", "||", "==", "!=, "<=" e ">=", respetivamente.

Feitas as devidas apresentações, o resultado é o seguinte:

T = {NAME, TEXT, EQLS, DIFF, SMEQ, GTEQ, NUMBER, AND, OR, START, END, WHILE, IF, READ, PRINT, ELSE, D1, D2, ';', '[', ']', '=', '(', ')', '{', '}', '+', '-', '*', '/', '%', ',', '<', '>', '$'}

NT = {Program, Declarations, Body, Statement, Prints, Else, Expression, Parcel, Factor, Value, Args, MoreArgs, BooleanExpression, BooleanFactor}

S = {Program}

P = {

Program → Declarations START Body END

Declarations → ϵ
Declarations → NAME ';'
Declarations → Declarations D1 NAME '[' NUMBER ']' ';'
Declarations → Declarations D2 NAME '[' NUMBER ']' '[' NUMBER ']' ';'

Body → ϵ
Body → Body Statement

Statement → NAME '=' Expression ';'
Statement → D1 NAME '[' Expression ']' '=' Expression ';'
Statement → D2 NAME '[' Expression ']' '[' Expression ']' '=' Expression ';'
Statement → WHILE '(' BooleanExpression ')' '' Body '' ';'
Statement → IF '(' BooleanExpression ')' '' Body '' Else ';'
Statement → READ '(' NAME ')' ';'
Statement → READ '(' D1 NAME '[' Expression ']' ')' ';'
Statement → READ '(' D2 NAME '[' Expression ']' '[' Expression ']' ')' ';'
Statement → PRINT '(' Prints ')' ';'
Statement → NAME '(' Args ')' ';'

Prints → Value
Prints → TEXT

Else → ϵ
Else → ELSE '' Body ''

Expression → Parcel

Expression → Expression '+' Parcel
Expression → Expression '-' Parcel

Parcel → Parcel '*' Factor
Parcel → Parcel '/' Factor
Parcel → Parcel '%' Factor
Parcel → Factor

Factor → NUMBER
Factor → Value
Factor → '(' Expression ')'

Value → NAME
Value → D1 NAME '[' Expression ']'
Value → D2 NAME '[' Expression ']' '[' Expression ']'

Args → ϵ
Args → MoreArgs

MoreArgs → Value
MoreArgs → NUMBER
MoreArgs → MoreArgs ',' NUMBER
MoreArgs → MoreArgs ',' Value

BooleanExpression → BooleanExpression AND BooleanFactor
BooleanExpression → BooleanExpression OR BooleanFactor
BooleanExpression → BooleanFactor

BooleanFactor → Expression EQLS Expression
BooleanFactor → Expression DIFF Expression
BooleanFactor → Expression '<' Expression
BooleanFactor → Expression '>' Expression
BooleanFactor → Expression SMEQ Expression
BooleanFactor → Expression GTEQ Expression
BooleanFactor → '(' BooleanExpression ')'
}

Chapter 3

Concepção/Desenho da Resolução

3.1 Estruturas de Dados

Dado não ser importante a implementação de uma estrutura de dados muito eficiente, para representar a tabela de identificadores das variáveis e guardar determinadas instruções, o grupo optou pela criação de listas ligadas, devido à sua simplicidade.

Primeiramente, começámos por criar a tabela de identificadores para guardar toda a informação relevante acerca das variáveis usadas, como o nome, o endereço na stack, o tipo e o número de colunas (para os casos do tipo ser um array, ou uma matriz). Esta estrutura revelar-se-á de extrema importância no futuro quando for necessário inicializar novas variáveis, pois irá verificar se já estão declaradas ou não e quando for preciso ir ao seu endereço buscar o valor guadado na stack.

```
typedef struct idtable {
        char *var;
        int type;
        int index;
        int columns;
        struct idtable *next;
} ID;

typedef struct idtable *IdTable;
```

De seguida, e para conseguir fazer o aninhamento de instruções de controlo de fluxo, criámos duas estruturas de dados (listas ligadas do tipo stack) que vão guardar temporáriamente o identificador da instrução *while* ou *if* em que o programa está no momento, para que, ao imprimir a instrução *assembly* de final de ciclo ou de *if*, possa relacioná-la com a instrução respetiva e denominá-la com o seu identificador.

```
typedef struct listwhiles {
        int numWhile;
        struct listwhiles *next;
} LWH;

typedef struct listwhiles *ListWhiles;

typedef struct listifs {
        int numIf;
        struct listifs *next;
} LIF;

typedef struct listifs *ListIfs;
```

3.2 Algoritmos Implementados

Para registar ou obter informação sobre as variáveis da tabela de identificadores, foram construídas algumas funções bastante simples:

- *insertVar*, que insere uma nova variável na tabela;
- *existVar*, verifica se uma determinada variável existe na tabela;
- *getTypeVar*, retorna o tipo de uma dada variável;
- *getVarIndex*, retorna o endereço da variável na stack;
- *getNumCols*, retorna o número de colunas de uma variável.

Chapter 4

Codificação

4.1 Alternativas, Decisões e Problemas de Implementação

Operadores

Um dos principais problemas passou pela definição da ordem com que são efetuadas as operações aritméticas. Neste caso, o grupo optou por, numa situação em que é detetada uma expressão, dividir a mesma em fatores e parcelas, ou seja, quando há uma soma ou subtração, poderá ocorrer uma expressão somar com uma parcela, que por sua vez, poderá ser um só valor, ou então uma multiplicação ou divisão de um fator com uma parcela. Isto irá garantir, com certezas absolutas, que os operadores com maior prioridade irão ser executados em primeiro lugar.

Aninhamento

Outro problema encontrado (e bem mais complexo que o anterior) foi o do aninhamento de instruções de controlo de fluxo, uma vez que será sempre necessário um retrocesso. Para dar a volta à situação, o grupo utilizou as duas stacks criadas (*ListIfs* e *ListWhiles*) e já explicadas anteriormente, para guardar o identificador da instrução *while* ou *if* em que o programa se encontra no momento. A título de exemplo, sempre que for detetado um *while*, o programa faz um *push* à stack com o seu identificador, ou seja, o primeiro que aparecer terá o número 1, o segundo o número 2, e assim sucessivamente. Quando chegar á última instrução, faz o *pop* à stack com o identificador atual, que está guardado em variável. Desta forma saber-se-à sempre a que *while* ou *if* correspondem as instruções atuais. Convém notar que, para evitar que duas *labels* de fim de instrução apareçam sucessivamente juntas no código *assembly* e comprometam o desempenho do programa, é também guardado o endereço da última instrução de fim de ciclo ou condição, para que, quando ocorrer um caso deste tipo, se possa acrescentar a instrução *nop*, que não faz absolutamente nada.

Conflitos

Ocorreu também uma situação em que, quando supostamente o programa deveria detetar uma variável como uma matriz, este detetou como sendo um array. Ora, isto ocorreu devido ao facto de as ações impostas no *Yacc* aquando da deteção de arrays e matrizes serem iguais no seu início e, por esse motivo, como a condição relativa ao array aparecia em primeiro lugar, o programa ia para lá e ignorava a condição das matrizes. Para resolver este problema, optámos por colocar um identificador com a dimensão do array/matriz (D1 e D2, respetivamente). Desta forma garante-se que as condições serão sempre diferentes e impede o aparecimento de conflitos.

4.2 Testes realizados e Resultados

De seguida, serão apresentados exemplos de código na linguagem proposta, assim como o respetivo *assembly* gerado para alguns dos testes propostos pela equipa docente para verificar o bom funcionamento do compilador:

Lidos 3 números, escrever o maior deles

```
x; y; z;

START

READ (x);
READ (y);
READ (z);


IF (x >= y) {

        IF (x >= z) {

                PRINT ("Resultado: ");
                PRINT (x);
        }

        ELSE {

                PRINT ("Resultado: ");
                PRINT (z);

        }
}
ELSE {

        IF (y >= z) {

                PRINT ("Resultado: ");
                PRINT (y);

        }
        ELSE {

                PRINT ("Resultado: ");
                PRINT (z);

        }

}

END
```

Assembly resultante:

```
1         pushi 0
2         pushi 0
3         pushi 0
4     start
5         read
6         atoi
7         storeg 0
8         read
9         atoi
10        storeg 1
11        read
12        atoi
13        storeg 2
14        pushg 0
15        pushg 1
16        supeq
17        jz else1
18        pushg 0
19        pushg 2
20        supeq
21        jz else2
22        pushs "Resultado: "
23        writes
24        pushg 0
25        writei
26        jump endif2
27    else2:
28        pushs "Resultado: "
29        writes
30        pushg 2
31        writei
32    endif2:
33        nop
34        jump endif1
35    else1:
36        pushg 1
37        pushg 2
38        supeq
39        jz else3
40        pushs "Resultado: "
41        writes
42        pushg 1
43        writei
44        jump endif3
45    else3:
46        pushs "Resultado: "
47        writes
48        pushg 2
49        writei
50    endif3:
51        nop
52    endif1:
53        nop
54        end
```

Ler N números, calcular e imprimir o seu somatório

```
1   i; n; x; sum;
2
3   START
4
5   PRINT ("How many numbers you want to read?\n");
6   READ (n);
7
8   WHILE (i < n) {
9
10          READ (x);
11
12          sum = sum + x;
13
14          i = i + 1;
15  }
16
17  PRINT ("Resultado: ");
18  PRINT (sum);
19
20  END
```

Assembly resultante:

```
1         pushi 0
2         pushi 0
3         pushi 0
4         pushi 0
5     start
6         pushs "How many numbers you want to read?\n"
7         writes
8         read
9         atoi
10        storeg 1
11    while1:
12        pushg 0
13        pushg 1
14        inf
15        jz endWhile1
16        read
17        atoi
18        storeg 2
19        pushg 3
20        pushg 2
21        add
22        storeg 3
23        pushg 0
24        pushi 1
25        add
26        storeg 0
27        jump while1
28    endWhile1:
29        pushs "Resultado: "
30        writes
31        pushg 3
32        writei
33        end
```

Contar e imprimir os números pares de uma sequência de N números dados

```
1   n; x; i; count;
2
3   START
4
5   PRINT ("How many numbers do you want to read?\n\n");
6   READ (n);
7
8   WHILE (i < n) {
9
10          READ (x);
11
12          IF (x % 2 == 0) {
13
14                  PRINT (x);
15                  count = count + 1;
16
17          }
18
19          i = i + 1;
20  }
21
22  PRINT ("\nNumber of pairs: ");
23  PRINT (count);
24
25  END
```

Assembly resultante:

```
1         pushi 0
2         pushi 0
3         pushi 0
4         pushi 0
5     start
6         pushs "How many numbers do you want to read?\n\n"
7         writes
8         read
9         atoi
10        storeg 0
11    while1:
12        pushg 2
13        pushg 0
14        inf
15        jz endWhile1
16        read
17        atoi
18        storeg 1
19        pushg 1
20        pushi 2
21        mod
22        pushi 0
23        equal
24        jz else1
25        pushg 1
26        writei
27        pushg 3
28        pushi 1
29        add
30        storeg 3
31    else1:
32        pushg 2
33        pushi 1
34        add
35        storeg 2
36        jump while1
37    endWhile1:
38        pushs "\nNumber of pairs: "
39        writes
40        pushg 3
41        writei
42        end
```

Ler e armazenar os elementos de um vetor de comprimento N. Imprimir os valores por ordem crescente após fazer a ordenação do array por trocas diretas

```
1   i; j; x; n; pos; swap; D1 array [3];
2
3   START
4
5   n = 3;
6
7   WHILE (i < n) {
8
9           READ (x);
10          D1 array [i] = x;
11          i = i + 1;
12  }
13
14  i = 0;
15
16  WHILE (i < n) {
17
18          pos = i;
19          j = i + 1;
20
21          WHILE (j < n) {
22
23                  IF (D1 array [pos] > D1 array [j]) {
24
25                          pos = j;
26
27                  }
28
29                  j = j + 1;
30
31          }
32
33          IF (pos != i) {
34
35                  swap = D1 array [i];
36                  D1 array [i] = D1 array [pos];
37                  D1 array [pos] = swap;
38
39          }
40
41          i = i + 1;
42
43  }
44
45  i = 0;
46
47  WHILE (i < n) {
48
49          PRINT (D1 array [i]);
50
51          IF (i != n - 1) {
52
53                  PRINT (", ");
54
55          }
56
57          i = i + 1;
58  }
59
60  END
```

Assembly resultante:

```
1         pushi 0
2         pushi 0
3         pushi 0
4         pushi 0
5         pushi 0
6         pushi 0
7         pushn 3
8     start
9         pushi 3
10        storeg 3
11    while1:
12        pushg 0
13        pushg 3
14        inf
15        jz endWhile1
16        read
17        atoi
18        storeg 2
19        pushgp
20        pushi 6
21        padd
22        pushg 0
23        pushg 2
24        storen
25        pushg 0
26        pushi 1
27        add
28        storeg 0
29        jump while1
30    endWhile1:
31        pushi 0
32        storeg 0
33    while2:
34        pushg 0
35        pushg 3
36        inf
37        jz endWhile2
38        pushg 0
39        storeg 4
40        pushg 0
41        pushi 1
42        add
43        storeg 1
44    while3:
45        pushg 1
46        pushg 3
47        inf
48        jz endWhile3
49        pushgp
50        pushi 6
51        padd
52        pushg 4
53        loadn
54        pushgp
55        pushi 6
56        padd
57        pushg 1
58        loadn
59        sup
60        jz else1
61        pushg 1
62        storeg 4
63    else1:
64        pushg 1
65        pushi 1
66        add
67        storeg 1
68        jump while3
69    endWhile3:
70        pushg 4
71        pushg 0
72        equal
73        pushi 0
74        equal
75        jz else2
76        pushgp
77        pushi 6
78        padd
```

```
79        pushg 0
80        loadn
81        storeg 5
82        pushgp
83        pushi 6
84        padd
85        pushg 0
86        pushgp
87        pushi 6
88        padd
89        pushg 4
90        loadn
91        storen
92        pushgp
93        pushi 6
94        padd
95        pushg 4
96        pushg 5
97        storen
98    else2:
99        pushg 0
100       pushi 1
101       add
102       storeg 0
103       jump while2
104   endWhile2:
105       pushi 0
106       storeg 0
107   while4:
108       pushg 0
109       pushg 3
110       inf
111       jz endWhile4
112       pushgp
113       pushi 6
114       padd
115       pushg 0
116       loadn
117       writei
118       pushg 0
119       pushg 3
120       pushi 1
121       sub
122       equal
123       pushi 0
124       equal
125       jz else3
126       pushs ", "
127       writes
128   else3:
129       pushg 0
130       pushi 1
131       add
132       storeg 0
133       jump while4
134   endWhile4:
135       nop
136       end
```

Ler e armazenar os elementos de uma matriz NxM. Calcular e imprimir a média e máximo dessa matriz

```
1   i; j; max; soma; med; x; n; m; D2 matrix [2][2];
2
3   START
4
5   n = 2;
6   m = 2;
7
8   WHILE (i < n) {
9
10          j = 0;
11
12          WHILE (j < m) {
13
14                  READ (x);
15
16                  D2 matrix [i][j] = x;
17
18                  soma = soma + x;
19
20                  IF (x > max) {
21
22                          max = x;
23
24                  }
25
26                  j = j + 1;
27
28          }
29
30          i = i + 1;
31
32  }
33
34  med = soma / (n * m);
35
36  PRINT ("Average = ");
37  PRINT (med);
38
39  PRINT ("\nMax = ");
40  PRINT (max);
41
42  END
```

Assembly resultante:

```
1         pushi 0
2         pushi 0
3         pushi 0
4         pushi 0
5         pushi 0
6         pushi 0
7         pushi 0
8         pushi 0
9         pushn 4
10    start
11        pushi 2
12        storeg 6
13        pushi 2
14        storeg 7
15    while1:
16        pushg 0
17        pushg 6
18        inf
19        jz endWhile1
20        pushi 0
21        storeg 1
22    while2:
23        pushg 1
24        pushg 7
25        inf
26        jz endWhile2
27        read
28        atoi
29        storeg 5
30        pushgp
31        pushi 8
32        padd
33        pushg 0
34        pushi 2
35        mul
36        pushg 1
37        add
38        pushg 5
39        storen
40        pushg 3
41        pushg 5
42        add
43        storeg 3
44        pushg 5
45        pushg 2
46        sup
47        jz else1
48        pushg 5
49        storeg 2
50    else1:
51        pushg 1
52        pushi 1
53        add
54        storeg 1
55        jump while2
56    endWhile2:
57        pushg 0
58        pushi 1
59        add
60        storeg 0
61        jump while1
62    endWhile1:
63        pushg 3
64        pushg 6
65        pushg 7
66        mul
67        div
68        storeg 4
69        pushs "Average = "
70        writes
71        pushg 4
72        writei
73        pushs "\nMax = "
74        writes
75        pushg 2
76        writei
77        end
```

Chapter 5

Conclusão

O grupo concluiu que o resultado final obtido é bom e os problemas encontrados foram todos ultrapassados. No que diz respeito á gramática desenvolvida pode-se concluir que cumpre todos os requisitos, incluindo o requisito opcional e ainda que a sua expansão será fácil em muitos aspectos, como por exemplo, a evolução para outros tipos, que não inteiros(float,doubles,etc).

Em relação ao compilador produzido e com base nos testes mínimos que foram desenvolvidos, este constrói efetivamente os ficheiros com as instruções assembly na ordem correta que permite ser interpretado pela máquina de stack virtual e ainda obter o resultado esperado na mesma.

Optou-se ainda por desenvolver outros testes que demonstram outras situações comuns em que o compilador também funciona.

Appendix A

Ficheiro *Yacc*

```
%{

#define _GNU_SOURCE

#define NO  0
#define YES 1

#define INTEGER 0
#define ARRAY   1
#define MATRIX  2

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

typedef struct idtable {
        char *var;              // Variable name
        int type;               // Variable type
        int index;              // Variable index on stack
        int columns;            // Number of columns (for integer is 1)
        struct idtable *next;
} ID;

typedef struct idtable *IdTable;

typedef struct listwhiles {
        int numWhile;
        struct listwhiles *next;
} LWH;

typedef struct listwhiles *ListWhiles;

typedef struct listifs {
        int numIf;
        struct listifs *next;
} LIF;

typedef struct listifs *ListIfs;

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

IdTable insertVar(IdTable t, char *var, int type, int index, int columns) {
        if(t == NULL) {
                t = (IdTable) malloc(sizeof(struct idtable));
                t->var = strdup(var);
                t->type = type;
                t->index = index;
                t->columns = columns;
                t->next = NULL;
        }
        else t->next = insertVar(t->next, var, type, index, columns);
        return t;
}

int existVar(IdTable t, char *var) {
        if(t == NULL) return NO;
```

```
58          else if(strcmp(var, t->var) == 0) return YES;
59          else return existVar(t->next, var);
60  }
61
62  int getTypeVar(IdTable t, char *var) {
63          if(t == NULL) return -1;
64          else if(strcmp(var, t->var) == 0) return t->type;
65          else return getTypeVar(t->next, var);
66  }
67
68  int getVarIndex(IdTable t, char *var) {
69          if(t == NULL) return -1;
70          else if(strcmp(var, t->var) == 0) return t->index;
71          else return getVarIndex(t->next, var);
72  }
73
74  int getNumCols(IdTable t, char *var) {
75          if(t == NULL) return 0;
76          else if(strcmp(var, t->var) == 0) return t->columns;
77          else return getNumCols(t->next, var);
78  }
79
80  FILE *fp;               // File where instructions will be saved
81
82  int sp = 0;             // Stack pointer
83  int addr = 0;           // Instruction index
84  int endAddr = 0;        // Index of the last flow control ending instruction
85  int numWhile = 0;       // Number of whiles
86  int numIf = 0;          // Number of ifs
87
88  IdTable t = NULL;
89  ListWhiles listWhiles = NULL;
90  ListIfs listIfs = NULL;
91
92  %}
93
94  %union {
95          int num;
96          char *str;
97  }
98
99  /* Terminal symbols */
100 %token <str> NAME TEXT EQLS DIFF SMEQ GTEQ
101 %token <num> NUMBER
102 %token AND OR START END WHILE IF READ PRINT ELSE D1 D2
103
104 %%
105
106 Program : Declarations {
107
108                 fprintf(fp, "start\n");
109                 addr++;
110
111         } START Body END {
112
113                 if(addr == endAddr + 1) fprintf(fp, "     nop\n     end\n");
114                 else fprintf(fp, "     end\n");
115                 fclose(fp);
116
117         }
118         ;
119
120 Declarations : /* EMPTY */
121              | Declarations NAME ';' {
122
123                         if(existVar(t, $2) == YES) {
124                                 char buf[50];
125                                 sprintf(buf, "variable %s already declared", $2);
126                                 yyerror(buf);
127                         }
128                         else {
129                                 t = insertVar(t, $2, INTEGER, sp, 1);
130                                 fprintf(fp, "     pushi 0\n");
131                                 sp++;
132                                 addr++;
133                         }
134
135              }
136              | Declarations D1 NAME '[' NUMBER ']' ';' {
137
138                         if(existVar(t, $3) == YES) {
```

```
139                                 char buf[50];
140                                 sprintf(buf, "variable %s already declared", $3);
141                                 yyerror(buf);
142                         }
143                         else {
144                                 t = insertVar(t, $3, ARRAY, sp, $5);
145                                 fprintf(fp, "     pushn %d\n", $5);
146                                 sp += $5;
147                                 addr++;
148                         }
149
150                 }
151                 | Declarations D2 NAME '[' NUMBER ']' '[' NUMBER ']' ';' {
152
153                         if(existVar(t, $3) == YES) {
154                                 char buf[50];
155                                 sprintf(buf, "variable %s already declared", $3);
156                                 yyerror(buf);
157                         }
158                         else {
159                                 t = insertVar(t, $3, MATRIX, sp, $8);
160                                 fprintf(fp, "     pushn %d\n", $5 * $8);
161                                 sp += $5 * $8;
162                                 addr++;
163                         }
164
165                 }
166                 ;
167
168     Body : /* EMPTY */
169          | Body Statement
170          ;
171
172     Statement : NAME '=' Expression ';' {
173
174                         if(existVar(t, $1) == YES) {
175                                 if(getTypeVar(t, $1) == INTEGER) {
176                                         int varIndex = getVarIndex(t, $1);
177                                         fprintf(fp, "     storeg %d\n", varIndex);
178                                         addr++;
179                                 }
180                                 else {
181                                         char buf[50];
182                                         sprintf(buf, "variable %s has another type", $1);
183                                         yyerror(buf);
184                                 }
185                         }
186                         else {
187                                 char buf[50];
188                                 sprintf(buf, "variable %s is not declared", $1);
189                                 yyerror(buf);
190                         }
191
192             }
193             | D1 NAME {
194
195                         if(existVar(t, $2) == YES) {
196                                 if(getTypeVar(t, $2) == ARRAY) {
197                                         int varIndex = getVarIndex(t, $2);
198                                         fprintf(fp, "     pushgp\n     pushi %d\n     padd\n", varIndex);
199                                         addr += 3;
200                                 }
201                                 else {
202                                         char buf[50];
203                                         sprintf(buf, "variable %s has another type", $2);
204                                         yyerror(buf);
205                                 }
206                         }
207                         else {
208                                 char buf[50];
209                                 sprintf(buf, "variable %s is not declared", $2);
210                                 yyerror(buf);
211                         }
212
213             } '[' Expression ']' '=' Expression ';' {
214
215                         fprintf(fp, "     storen\n");
216                         addr++;
217
218             }
219             | D2 NAME {
```

```
220
221                     if(existVar(t, $2) == YES) {
222                             if(getTypeVar(t, $2) == MATRIX) {
223                                     int varIndex = getVarIndex(t, $2);
224                                     fprintf(fp, "     pushgp\n     pushi %d\n     padd\n", varIndex);
225                                     addr += 3;
226                             }
227                             else {
228                                     char buf[50];
229                                     sprintf(buf, "variable %s has another type", $2);
230                                     yyerror(buf);
231                             }
232                     }
233                     else {
234                             char buf[50];
235                             sprintf(buf, "variable %s is not declared", $2);
236                             yyerror(buf);
237                     }
238
239             } '[' Expression ']' {
240
241                     int numCols = getNumCols(t, $2);
242                     fprintf(fp, "     pushi %d\n     mul\n", numCols);
243                     addr += 2;
244
245             } '[' Expression ']' '=' {
246
247                     fprintf(fp, "     add\n");
248                     addr++;
249
250             } Expression ';' {
251
252                     fprintf(fp, "     storen\n");
253                     addr++;
254
255             }
256             | WHILE {
257
258                     numWhile++;
259                     ListWhiles aux = (ListWhiles) malloc(sizeof(struct listwhiles));
260                     aux->numWhile = numWhile;
261                     aux->next = listWhiles;
262                     listWhiles = aux;
263                     fprintf(fp, "while%d:\n", numWhile);
264                     addr++;
265
266             } '(' BooleanExpression ')' {
267
268                     fprintf(fp, "     jz endWhile%d\n", numWhile);
269                     addr++;
270
271             } '{' Body '}' {
272
273                     fprintf(fp, "     jump while%d\n", listWhiles->numWhile);
274                     addr++;
275                     if(endAddr == addr - 1) {
276                             fprintf(fp, "     nop\nendWhile%d:\n", listWhiles->numWhile);
277                             addr++;
278                     }
279                     else fprintf(fp, "endWhile%d:\n", listWhiles->numWhile);
280                     endAddr = addr;
281                     addr++;
282                     listWhiles = listWhiles->next;
283
284             }
285             | IF '(' BooleanExpression ')' {
286
287                     numIf++;
288                     ListIfs aux = (ListIfs) malloc(sizeof(struct listifs));
289                     aux->numIf = numIf;
290                     aux->next = listIfs;
291                     listIfs = aux;
292                     fprintf(fp, "     jz else%d\n", numIf);
293                     addr++;
294
295             } '{' Body '}' Else
296             | READ '(' NAME ')' ';' {
297
298                     if(existVar(t, $3) == YES) {
299                             if(getTypeVar(t, $3) == INTEGER) {
300                                     int varIndex = getVarIndex(t, $3);
```

```
301                                     fprintf(fp, "     read\n     atoi\n     storeg %d\n", varIndex);
302                                     addr += 3;
303                             }
304                             else {
305                                     char buf[50];
306                                     sprintf(buf, "variable %s has another type", $3);
307                                     yyerror(buf);
308                             }
309                     }
310                     else {
311                             char buf[50];
312                             sprintf(buf, "variable %s is not declared", $3);
313                             yyerror(buf);
314                     }
315
316             }
317             | READ '(' D1 NAME {
318
319                     if(existVar(t, $4) == YES) {
320                             if(getTypeVar(t, $4) == ARRAY) {
321                                     int varIndex = getVarIndex(t, $4);
322                                     fprintf(fp, "     pushgp\n     pushi %d\n     padd\n", varIndex);
323                                     addr += 3;
324                             }
325                             else {
326                                     char buf[50];
327                                     sprintf(buf, "variable %s has another type", $4);
328                                     yyerror(buf);
329                             }
330                     }
331                     else {
332                             char buf[50];
333                             sprintf(buf, "variable %s is not declared", $4);
334                             yyerror(buf);
335                     }
336
337             } '[' Expression ']' ')' ';' {
338
339                     fprintf(fp, "     read\n     atoi\n     storen\n");
340                     addr += 3;
341
342             }
343             | READ '(' D2 NAME {
344
345                     if(existVar(t, $4) == YES) {
346                             if(getTypeVar(t, $4) == MATRIX) {
347                                     int varIndex = getVarIndex(t, $4);
348                                     fprintf(fp, "     pushgp\n     pushi %d\n     padd\n", varIndex);
349                                     addr += 3;
350                             }
351                             else {
352                                     char buf[50];
353                                     sprintf(buf, "variable %s has another type", $4);
354                                     yyerror(buf);
355                             }
356                     }
357                     else {
358                             char buf[50];
359                             sprintf(buf, "variable %s is not declared", $4);
360                             yyerror(buf);
361                     }
362
363             } '[' Expression ']' {
364
365                     int numCols = getNumCols(t, $4);
366                     fprintf(fp, "     pushi %d\n     mul\n", numCols);
367                     addr += 2;
368
369             } '[' Expression ']' ')' ';' {
370
371                     fprintf(fp, "     add\n     read\n     atoi\n     storen\n");
372                     addr += 4;
373
374             }
375             | PRINT '(' Prints ')' ';'
376             | NAME '(' Args ')' ';' {
377
378                     fprintf(fp, "     pusha %s\n     call\nretorno:\n     nop", $1);
379                     addr++;
380
381             }
```

```
382             ;
383
384     Prints : Value {
385
386                             fprintf(fp, "     writei\n");
387                             addr++;
388
389             }
390             | TEXT {
391
392                             fprintf(fp, "     pushs %s\n     writes\n", $1);
393                             addr += 2;
394
395             }
396             ;
397
398     Else : /* EMPTY */ {
399
400                             if(endAddr == addr - 1) {
401                                     fprintf(fp, "     nop\nelse%d:\n", listIfs->numIf);
402                                     addr++;
403                             }
404                             else fprintf(fp, "else%d:\n", listIfs->numIf);
405                             listIfs = listIfs->next;
406                             endAddr = addr;
407                             addr++;
408
409         }
410         | ELSE {
411
412                             if(endAddr == addr - 1) {
413                                     fprintf(fp, "     nop\n     jump endif%d\nelse%d:\n", listIfs->numIf, listIfs->
                                            numIf);
414                                     addr++;
415                             }
416                             else fprintf(fp, "     jump endif%d\nelse%d:\n", listIfs->numIf, listIfs->numIf);
417                             endAddr = addr;
418                             addr++;
419
420         } '{' Body '}' {
421
422                             if(endAddr == addr - 1) {
423                                     fprintf(fp, "     nop\nendif%d:\n", listIfs->numIf);
424                                     addr++;
425                             }
426                             else fprintf(fp, "endif%d:\n", listIfs->numIf);
427                             listIfs = listIfs->next;
428                             endAddr = addr;
429                             addr++;
430
431         }
432         ;
433
434     Expression : Parcel
435                | Expression '+' Parcel {
436
437                             fprintf(fp, "     add\n");
438                             addr++;
439
440                }
441                | Expression '-' Parcel {
442
443                             fprintf(fp, "     sub\n");
444                             addr++;
445
446                }
447                ;
448
449     Parcel : Parcel '*' Factor {
450
451                             fprintf(fp, "     mul\n");
452                             addr++;
453
454             }
455             | Parcel '/' Factor {
456
457                             fprintf(fp, "     div\n");
458                             addr++;
459
460             }
461             | Parcel '%' Factor {
```

```
462
463                             fprintf(fp, "     mod\n");
464                             addr++;
465
466             }
467             | Factor
468             ;
469
470     Factor : NUMBER {
471
472                             fprintf(fp, "     pushi %d\n", $1);
473                             addr++;
474
475             }
476             | Value
477             | '(' Expression ')'
478             ;
479
480     Value : NAME {
481
482                             if(existVar(t, $1) == YES) {
483                                     if(getTypeVar(t, $1) == INTEGER) {
484                                             int varIndex = getVarIndex(t, $1);
485                                             fprintf(fp, "     pushg %d\n", varIndex);
486                                             addr++;
487                                     }
488                                     else {
489                                             char buf[50];
490                                             sprintf(buf, "variable %s has another type", $1);
491                                             yyerror(buf);
492                                     }
493                             }
494                             else {
495                                     char buf[50];
496                                     sprintf(buf, "variable %s is not declared", $1);
497                                     yyerror(buf);
498                             }
499
500             }
501             | D1 NAME {
502
503                             if(existVar(t, $2) == YES) {
504                                     if(getTypeVar(t, $2) == ARRAY) {
505                                             int varIndex = getVarIndex(t, $2);
506                                             fprintf(fp, "     pushgp\n     pushi %d\n     padd\n", varIndex);
507                                             addr += 3;
508                                     }
509                                     else {
510                                             char buf[50];
511                                             sprintf(buf, "variable %s has another type", $2);
512                                             yyerror(buf);
513                                     }
514                             }
515                             else {
516                                     char buf[50];
517                                     sprintf(buf, "variable %s is not declared", $2);
518                                     yyerror(buf);
519                             }
520
521             } '[' Expression ']' {
522
523                             fprintf(fp, "     loadn\n");
524                             addr++;
525
526             }
527             | D2 NAME {
528
529                             if(existVar(t, $2) == YES) {
530                                     if(getTypeVar(t, $2) == MATRIX) {
531                                             int varIndex = getVarIndex(t, $2);
532                                             fprintf(fp, "     pushgp\n     pushi %d\n     padd\n", varIndex);
533                                             addr += 3;
534                                     }
535                                     else {
536                                             char buf[50];
537                                             sprintf(buf, "variable %s has another type", $2);
538                                             yyerror(buf);
539                                     }
540                             }
541                             else {
542                                     char buf[50];
```

```
543                                 sprintf(buf, "variable %s is not declared", $2);
544                                 yyerror(buf);
545                         }
546
547         } '[' Expression ']' {
548
549                         int numCols = getNumCols(t, $2);
550                         fprintf(fp, "     pushi %d\n     mul\n", numCols);
551                         addr += 2;
552
553         } '[' Expression ']' {
554
555                         fprintf(fp, "     add\n     loadn\n");
556                         addr += 2;
557
558         }
559         ;
560
561     Args : /* EMPTY */
562          | MoreArgs
563          ;
564
565     MoreArgs : Value
566              | NUMBER {
567
568                         fprintf(fp, "     pushi %d\n", $1);
569                         addr++;
570
571              }
572              | MoreArgs ',' NUMBER {
573
574                         fprintf(fp, "     pushi %d\n", $3);
575                         addr++;
576
577              }
578              | MoreArgs ',' Value
579              ;
580
581     BooleanExpression : BooleanExpression AND BooleanFactor {
582
583                         fprintf(fp, "     mul\n");
584                         addr++;
585
586                   }
587                   | BooleanExpression OR BooleanFactor {
588
589                         fprintf(fp, "     add\n");
590                         addr++;
591
592                   }
593                   | BooleanFactor
594                   ;
595
596     BooleanFactor : Expression EQLS Expression {
597
598                         fprintf(fp, "     equal\n");
599                         addr++;
600
601               }
602               | Expression DIFF Expression {
603
604                         fprintf(fp, "     equal\n     pushi 0\n     equal\n");
605                         addr += 3;
606
607               }
608               | Expression '<' Expression {
609
610                         fprintf(fp, "     inf\n");
611                         addr++;
612
613               }
614               | Expression '>' Expression {
615
616                         fprintf(fp, "     sup\n");
617                         addr++;
618
619               }
620               | Expression SMEQ Expression {
621
622                         fprintf(fp, "     infeq\n");
623                         addr++;
```

```
624
625                 }
626                 | Expression GTEQ Expression {
627
628                         fprintf(fp, "     supeq\n");
629                         addr++;
630
631                 }
632                 | '(' BooleanExpression ')'
633                 ;
634
635     %%
636
637     #include "lex.yy.c"
638
639     int yyerror(char *s) {
640             fprintf(stderr, "%s (line %d)\n", s, yylineno);
641             return 0;
642     }
643
644     int main() {
645             fp = fopen("test.vm", "w");
646             yyparse();
647             return 0;
648     }
```

Appendix B

Ficheiro *Flex*

```
%option noyywrap
%option yylineno

inteiro         [0-9]+
name            [a-z_]+
text            \"[^\"]*\"
and             \&\&
or              \|\|
equals          \=\=
different       \!\=
smallerEqual    \<\=
greaterEqual    \>\=
symbols         [;\[\]={},\(\)\+\-*\/%<>]

%%

{inteiro}                       {yylval.num = atoi(yytext); return NUMBER;}
{name}                          {yylval.str = strdup(yytext); return NAME;}
{text}                          {yylval.str = strdup(yytext); return TEXT;}

START                           {return START;}
END                             {return END;}
WHILE                           {return WHILE;}
IF                              {return IF;}
READ                            {return READ;}
PRINT                           {return PRINT;}
ELSE                            {return ELSE;}
D1                              {return D1;}
D2                              {return D2;}

{and}                           {yylval.str = strdup(yytext); return AND;}
{or}                            {yylval.str = strdup(yytext); return OR;}
{equals}                        {yylval.str = strdup(yytext); return EQLS;}
{different}                     {yylval.str = strdup(yytext); return DIFF;}
{smallerEqual}                  {yylval.str = strdup(yytext); return SMEQ;}
{greaterEqual}                  {yylval.str = strdup(yytext); return GTEQ;}

{symbols}                       {return(yytext[0]);}
[ \n\t]                         {}
.                               {yyerror("Unknown character");}

%%
```

Appendix C

Makefile

```
all: compiler

compiler: compiler.fl compiler.y
        flex compiler.fl
        yacc compiler.y
        gcc y.tab.c -o compiler
```

Bibliography

[1] *V. Aho, Alfred* , *S. Lam,Monica* , *Sethi,Ravi* and *D. Ullman,Jeffrey* Second Edition, *Compilers Principles Techniques and Tools.*