Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 3 - YACC

Relatório de Desenvolvimento

Alexandre Lopes Mandim da Silva
A73674

Luís Miguel da Cunha Lima
A74260

Hugo Alves Carvalho
A74219

12 de Junho de 2017

Resumo

Este relatório descreve todo o processo de desenvolvimento e decisões tomadas para a realização do trabalho prático nº 3 da Unidade Curicular de Processamento de Linguagens.

O problema a resolver consiste no desenvolvimento de uma linguagem de programação imperativa capaz de satisfazer determinados requisitos.

Conteúdo

Capítulo 1

Problema

1.1 Introdução

Este projeto tem como principais objetivos aumentar a experiência nos campos da engenharia de linguagens e programação generativa, através do desenvolvimento de linguagens e a aplicação de geradores de compiladores apoiados em gramáticas tradutoras como o Yacc.

Este projeto possui ainda como objetivos secundários aprofundar a capacidade de desenvolvimento de gramáticas independentes de contexto, e melhorar o uso do ambiente Linux e da linguagem imperativa C.

O actual relatório tem como objetivo esclarecer o processo de desenvolvimento de uma linguagem de programação imperativa, e o seu respetivo compilador, que deve ser competente de gerar pseudo-código Assembly da máquina virtual VM, utilizada neste projeto. Para isto, é importante conceber uma gramática independente de contexto que defina a linguagem, e estabeleça as normas de tradução para o Assembly da VM.

1.2 Enunciado

Pretende-se que comece por definir uma linguagem de programação imperativa simples, a seu gosto.

Apenas deve ter em consideração que essa linguagem terá de permitir:

- declarar e manusear variáveis atómicas do tipo inteiro, com os quais se podem realizar as habituais operações aritméticas, relacionais e lógicas.
- declarar e manusear variáveis estruturadas do tipo array (a 1 ou 2 dimensões) de inteiros, em relação aos quais é apenas permitida a operação de indexação (índice inteiro).
- efetuar instruções algorítmicas básicas como a atribuição de expressões a variáveis.
- ler do standard input e escrever no standard output.
- efetuar instruções para controlo do fluxo de execuçãocondicional e cíclicaque possam ser aninhadas.
- definir e invocar subprogramas sem parâmetros, mas que possam retornar um resultado atómico (opcional).

Como é da praxe neste tipo de linguagens, as variáveis deverão ser declaradas no início do programa e não pode haver redeclarações, nem utilizações sem declaração prévia. Se nada for explicitado, o valor da variável após a declaração é 0 (zero).

Desenvolva, então, um compilador para essa linguagem com base na GIC criada acima e com recurso ao Gerador Yacc/Flex.

O compilador deve gerar pseudo-código, Assembler da Máquina Virtual VM cuja documentação completa está disponibilizada no Bb.

Muito importante:

Para a entrega do TP deve preparar um conjunto de testes (programas-fonte escritos na sua linguagem) e mostrar o código Assembly gerado bem como o programa a correr na máquina virtual VM. Esse conjunto terá de conter, no mínimo, os 6 exemplos que se seguem:

- ler 4 números e dizer se podem ser os lados de um quadrado.
- ler um inteiro N, depois ler N números e escrever o menor deles.
- ler N (constante do programa) números e calcular e imprimir o seu produtório.
- contar e imprimir os números impares de uma sequência de números naturais.
- ler e armazenar os elementos de um vetor de comprimento N; imprimir os valores por ordem decrescente após fazer a ordenação do array por trocas diretas.
- ler e armazenar N números num array; imprimir os valores por ordem inversa.

1.3 Estrutura do relátorio

A elaboração deste relatório segue a estrutura fornecida pelo docente.

O relátorio encontra-se dividido em quatro capítulos. O primeiro capítulo faz uma breve contextualização ao tema abordado neste relátorio. Posteriormente são desenvolvidas e justificadas todas as tarefas e raciocinios para o desenvolvimento dos objetivos deste trabalho prático. Para finalizar, faz-se uma análise crítica referente ao desenvolvimento do projeto e ao seu estado final.

Capítulo 2

Conceção e desenvolvimento da linguagem

2.1 Desenho da Linguagem

Para dar resposta as funcionalidades descritas no enunciado desenvolvemos uma linguagem de programação imperativa simples. Em seguida segue-se a sua explicação e alguns exemplos:

O start symbol é 'Programa' em que um 'Programa' pode ser uma ou mais funções.

```
Programa:     Funcoes                    {;}
            ;
Funcoes:      Funcao
            | Funcoes Funcao             {;}
            ;
```

Uma 'Função' começa com a letra 'F', seguido do nome da função e ''. Cada 'Funcao' é composta por 'Declaracoes' e 'Instrucoes' terminando com o caracter ''.

```
F nomeFuncao{

    INT a;

    a = 1;
}
```

A linguagem deve permitir declarar e manusear variáveis do tipo inteiro, bool e arrays de inteiros de uma ou duas dimensões. Para declarar uma variável de algum destes tipos é necessário escrever o seu tipo ('INT', 'BOOL', 'INT []', 'INT [][]') seguido de uma variável composta unicamente por letras finalizando com o caracter ';'. Aliás, qualquer declaração ou instrução deve terminar sempre com este caracter.

```
F nomeFuncao{

    INT inteiro;
    INT [10] array;
    INT [10][5] matrizInteiros;
    BOOL verdadeiro;
}
```

Com todas as 'Declaracoes' realizadas seguem-se as 'Instrucoes'. Estas podem ser atribuições de valores às variáveis previamente declaradas, instruções de I/O, instruções de controlo de fluxo de execução condicional e cíclica. Para realizar atribuições a variáveis é necessário escrever a variável (caso seja um array ou uma matriz os respetivos índices) seguido do caracter '=' e de uma operação ou de um valor, caso seja um inteiro array ou matriz essa operação

é aritmética (soma, subtração, divisão, multiplicação e módulo), por outro lado, se for uma variável do tipo bool essa operação é lógica ou relacional.

```
inteiro = 5;
[0] array = 10 + 5;
[inteiro][2] matrizInteiros = inteiro * [0] array;
verdadeiro = 5 SUP 1;
```

Para realizarmos instruções de controlo de fluxo condicional, usamos a palavra 'IF' seguido de uma operação lógica ou relacional entre parêntesis, que vai ditar se executa o bloco de instruções que vem em seguida e que esta entre chavetas. Caso a operação seja falsa, são executadas as instruções que se encontram no bloco else. Este bloco vem em seguida da última chaveta do 'IF' e começa com a palavra 'ELSE'. De salientar que podemos realizar estas instruções de controlo de forma aninhada.

```
IF([ j ] vetor INF [ min ] vetor){
    min = j;
}
ELSE{}
j = j + 1;
```

Para controlar o fluxo de execução de maneira cíclica é usada a instrução 'WHILE'. Está instrução, tal como a instrução 'IF' acompanha-se de uma operação logica ou relacional que vai impor a execução cíclica das instruções que se encontram em seguida entre chavetas. Tal como na situação de 'IF' 'ELSE', também podemos ter vários 'WHILE' aninhados.

```
WHILE(i SUPEQUAL 0){
    write [ i ] vetor;
    i = i - 1;
}
```

Por fim, temos as instruções de I/O. Para realizarmos input usamos a palavra 'READ' e respetiva é idêntica a uma atribuição. A instrução de output é iniciada pela palavra 'WRITE' e a variável que queremos imprimir.

```
F IO{

    INT input;

    input = read;
    write input;
}
```

2.2 Gramática

Neste tópico vamos explicar a gramatica independente de contexto usada para satisfazer as necessidades desta linguagem.

A nossa gramática começa com o não terminal 'PROGRAMA' que deriva em 'Funcoes' para permitir que um programa seja composto por uma ou mais 'Funcoes'. Cada função deriva em 'Declaracoes' e 'Instrucoes'. Cada 'Declaracao' é composta pelos terminais que podemos ver na imagem em baixo.

Uma 'Instrucao' pode derivar em quatro tipos: 'Atribuicao', 'I0', 'SE' e 'CICLO'. Uma 'Atribuicao' começa com os símbolos terminais que representam a variável a ser atribuída acompanhados com o símbolo que representa a atribuição '='. Cada 'Atribuicao' é também composta por um dos seguintes símbolos não terminais: 'Op' e 'OpLogAO'.

```
Programa:     Funcoes
            ;
Funcoes:      Funcao
            | Funcoes Funcao
            ;
Funcao:       F id '{' Declaracoes  Instrucoes '}'
            ;
Declaracoes:  Declaracao
            | Declaracoes Declaracao
            ;
Declaracao:   INT id ';'
            | INT '[' num ']' id ';'
            | INT '[' num ']' '[' num ']' id ';'
            | BOOL id ';'
            ;
Instrucoes:   /* empty */
            | Instrucoes Instrucao
            ;
Instrucao:    Atribuicao ';'
            | IO ';'
            | SE
            | CICLO
            ;
CICLO:        WHILE '('
                OpLogAO
                    ')' '{' Instrucoes
                    '}'
            ;
SE:           IF '(' OpLogAO
                ')' '{' Instrucoes
                '}' SENAO
            ;
SENAO:        ELSE '{'
                Instrucoes
                '}'
            ;
IO:           WRITE Termo
            ;

Atribuicao:   id '=' Op
            | id '=' OpLogAO
            | '[' num ']' id '=' Op
            | '[' id ']' id '='
                Op
            | '[' num ']' '[' num ']' id '=' Op
            | '[' num ']' '[' id ']' id '=' Op
            | '[' id ']' '[' num ']' id '=' Op
            | '[' id ']' '[' id ']' id '='  Op
            ;
Op:           READ
            | OpArit
            ;

OpArit:       Termo
            | OpArit '+' Termo
            | OpArit '-' Termo
            | OpArit '*' Termo
            | OpArit '/' Termo
            | OpArit '%' Termo
            ;
OpLogAO     : OpLog
            | OpLogAO AND OpLog
            | OpLogAO OR OpLog
            ;
OpLog:        Termo
            | Termo EQUAL Termo
            | Termo DIFFERENT Termo
            | Termo SUP Termo
            | Termo INF Termo
            | Termo SUPEQUAL Termo
            | Termo INFEQUAL Termo
            ;
Termo:        CONST
            | ID
            ;
```

Estes símbolos não terminais vao representar operações aritméticas, lógicas e relacionais, bem como operação de input. Estes terminais são explicados em seguida:

O terminal 'Op' representa uma operação aritmética('OpArit') ou o símbolo terminal 'READ' (usado para operação de input). Tal como podemos visualizar na figura em cima uma operação aritmética é composta por 'Termo' que representa uma variável ou uma constante, um símbolo terminal que representa a operação ('+', '-', '*', '/' , '%') e outra 'OpArit' para permitir recursividade do tipo (a + 5 * 10). O símbolo não terminal 'OpLogAO' segue o mesmo raciocínio que o 'OpArit' contudo apenas é usado para operações logicas e relacionais ('AND', 'OR', 'EQUAL', 'DIFFERENT', 'SUP', 'INF', 'SUPEQUAL', 'INFEQUAL').

Uma instrução deriva também em 'IO'. Este símbolo não terminal representa o input e é derivado no símbolo terminal 'WRITE' e no não terminal 'Termo'. Deriva também no símbolo não terminal 'SE' que representa o controlo de fluxo de execução condicional. Este símbolo deriva no terminal 'IF', '(', 'OpLogAO' (explicado no parágrafo anterior), ')', '{', 'Instrucoes', '}' e 'SENAO'. Este último representa as instruções a serem executadas caso a condição do 'IF' não se verifique.

Por fim, uma 'Instrucao' pode derivar no símbolo não terminal 'CICLO'. Tal como no 'SE' este deriva no não terminal 'WHILE' e '(' seguido de 'OpLogAO' (condição para paragem de ciclo), ')', '{', 'Instrucoes', '}'.

2.3 Ações Semânticas

As ações semânticas são o código C que é executado quando as produções são reconhecidas. Nestas ações vão ser executadas essencialmente três tipos de instruções: *printf's* com instruções pseudocódigo máquina, operações com estruturas de dados e verificação de erros.

O principal objetivo das ações semânticas é a criação de pseudocódigo máquina para que a máquina virtual possa executar. Por exemplo, quando variáveis são declaradas, realizamos *printf's* de *push* para alocar memória na stack para estas variáveis. Sempre que uma produção de atribuição, controlo de fluxo de execução condicional e cíclica, instruções de *input/output* e restantes instruções vão ser executados *printf's* do código máquina de modo a realizar o pretendido em cada instrução.

Sempre que uma variável é declarada, é necessário guardar a informação relativa a essa variável. Por exemplo, sempre que queremos fazer *push* dessa variável, precisamos de saber o seu endereço, ou se existe já uma variável com esse nome. Por outro lado, sempre que existir um 'IF' ou um 'WHILE' é necessário guardar a informação acerca destes para auxiliar a construção do código máquina. Deste modo, é necessário guardar informação relativa a variáveis e instruções de controlo de fluxo.

Finalmente, é também executado funções auxiliares de verificação de erros. Por exemplo, quando realizamos a atribuição de uma variável do tipo inteiro, esta espera um valor inteiro e não booleano.

2.4 Estruturas de Dados

Tal como explicado nas ações semânticas, é necessário guardar a informação sobre variáveis e instruções de controlo de fluxo. Assim sendo, usamos dois tipos de estruturas de dados: uma *hashtable* composta por elementos do tipo Variável (que representam uma variável) e duas listas ligadas que representam *stacks* para as instruções 'IF' e 'WHILE'.

```
typedef struct variavel{
    char *nome, *tipo;
    int nrLinha;
    int nrColuna;
    int endereco;
}*Variavel;
```

Para a *hashtable* usamos a biblioteca "search.h" e a *struct* variável em que guardamos o nome e tipo da variável, caso seja um *array* ou matriz guardamos o número de linhas e número de coluna, bem como o seu respetivo endereço na stack.

Como ver na imagem em baixo, é criada uma *hashtable* com 500 entradas e no final é libertada a memória.

Foram também criadas duas funções, "addVariavel" e "getVariavel" que adiciona e retorna uma variável da *hashtable*, respetivamente.

```
int main () {
    hcreate(500);
    yyparse();
    hdestroy();
    return 0;
}

typedef struct item{
    int nr;
    struct item *downIF;
} *Item;
```

Finalmente foi criada uma lista ligada que representa uma *stack* em que cada item contém um número que representa o "id" de um determinado 'IF' ou 'CICLO' e um apontador para o item em cima. Esta estrutura é importante pois quando temos instruções para controlo de fluxo aninhadas, sempre que saímos de um tipo de instrução precisamos de saber dados sobre a instrução anterior.

Capítulo 3

Testes e Exemplos de Utilização

3.1 Testes

Tal como pedido no enunciado, foi preparado um conjunto de testes para demonstrar a funcionalidade da linguagem.

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

  Código da nossa linguagem:

```
F Quadrado {
    INT [4] nrs;
    INT aux;
    INT prox;
    BOOL r;

    aux = 0;
    prox = aux + 1;

    WHILE (aux INF 4){
        [ aux ] nrs = read;
        aux = aux + 1;
    }
    IF([0] nrs SUP 0 AND [1] nrs SUP 0 AND [2] nrs SUP 0 AND [3] nrs SUP 0){
        IF ([0] nrs EQUAL [1] nrs AND [1] nrs EQUAL [2] nrs AND [2] nrs EQUAL [3] nrs){
            WRITE TRUE ;
        }
        ELSE{
            WRITE FALSE ;
        }
    }
    ELSE{
        WRITE FALSE;
    }
}
```

  Código máquina respectivo:

```
start
pushn 4
pushi 0
pushi 0
pushi 0
pushi 0
storel 4
pushl 4
pushi 1
add
storel 5
lblwhile1: nop
pushl 4
pushi 4
inf
jz lblfimwhile1
pushfp
pushi 0
padd
pushl 4
read
atoi
storen
pushl 4
pushi 1
add
storel 4
jump lblwhile1
lblfimwhile1: nop
pushl 0

    pushi 0
    sup
    pushl 1
    pushi 0
    sup
    mul
    pushl 2
    pushi 0
    sup
    mul
    pushl 3
    pushi 0
    sup
    mul
    lblif1: nop
    jz lblelse1
    pushl 0
    pushl 1
    equal
    pushl 1
    pushl 2
    equal
    mul
    pushl 2
    pushl 3
    equal
    mul
    lblif2: nop
    jz lblelse2
    pushi 1
```

```
writei
jump lblelseend2
lblifend2: nop
lblelse2: nop
pushi 0
writei
lblelseend2: nop
jump lblelseend1
lblifend1: nop
lblelse1: nop
pushi 0
writei
lblelseend1: nop
stop
```

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

  Código da nossa linguagem:

```
F Menor {
    INT n;
    INT aux;
    INT menor;
    INT ler;

    n = read;
    aux = 0;
    menor = 0;

    while(aux INF n){
        ler = read;

        if(menor SUP ler OR aux EQUAL 0){
            menor = ler;
        }
        else{}
        aux = aux + 1;
    }

    write menor;
}
```

  Código máquina respectivo:

```
start
pushi 0
pushi 0
pushi 0
pushi 0
pushi 0
storel 3
pushi 3
storel 0
pushi 1
storel 1
lblwhile1: nop
pushl 3
pushl 0
inf
jz lblfimwhile1
read
atoi
storel 2
pushl 1
pushl 2
mul
storel 1
pushl 3
pushi 1
add
storel 3
jump lblwhile1
lblfimwhile1: nop
pushl 1
writei
stop
```

```
 jz lblelse1
 pushl 3
 storel 2
 jump lblelseend1
 lblifend1: nop
 lblelse1: nop
 lblelseend1: nop
 pushl 1
 pushi 1
 add
 storel 1
 jump lblwhile1
 lblfimwhile1: nop
 pushl 2
 writei
 stop
```

- Ler N números e calcular o seu produtório

  Código da nossa linguagem:

```
F Produtorio {
    INT constante;
    INT produtorio;
    INT ler;
    INT aux;

    aux = 0;
    constante = 3;
    produtorio = 1;

    while(aux INF constante){
        ler = read;
        produtorio = produtorio * ler;
        aux = aux + 1;
    }

    write produtorio;
}
```

Código máquina respectivo:

```
start
pushi 0
pushi 0
pushi 0
pushi 0
pushi 0
storel 3
pushi 3
storel 0
pushi 1
storel 1
lblwhile1: nop
pushl 3
pushl 0
inf
jz lblfimwhile1
read
atoi
storel 2
pushl 1
pushl 2
mul
storel 1
pushl 3
pushi 1
add
storel 3
jump lblwhile1
lblfimwhile1: nop
pushl 1
writei
stop
```

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

  Código da nossa linguagem:

```
F Impares {
    INT n;
    INT [5] vetor;
    INT ler;
    INT aux;
    INT mod;
    INT contar;

    n = 5;
    contar = 0;

    aux = 0;
    while( aux inf n){
        [ aux ] vetor = read;
        aux = aux + 1;
    }

    aux = 0;
    while( aux inf n){
        mod = [ aux ] vetor % 2;
        if(mod EQUAL 1){
            write [ aux ] vetor ;
            contar = contar + 1 ;
        }
        else{}
        aux = aux + 1;
    }
    write contar;
}
```

Código máquina respectivo:

```
start
pushi 0
pushn 5
pushi 0
pushi 0
pushi 0
pushi 0
pushi 5
storel 0
pushi 0
storel 9
pushi 0
storel 7
lblwhile1: nop
pushl 7
pushl 0
inf
jz lblfimwhile1
pushfp
pushi 1
padd
pushl 7
read
atoi
storen
pushl 7
pushi 1
add
storel 7
jump lblwhile1
lblfimwhile1: nop
pushi 0
```

```
storel 7
lblwhile2: nop
pushl 7
pushl 0
inf
jz lblfimwhile2
pushfp
pushi 1
padd
pushl 7
loadn
pushi 2
mod
storel 8
pushl 8
pushi 1
equal
lblif1: nop
jz lblelse1
pushfp
pushi 1
padd
pushl 7
loadn
writei
pushl 9
pushi 1
add
storel 9
jump lblelseend1
lblifend1: nop
lblelse1: nop

 lblelseend1: nop
 pushl 7
 pushi 1
 add
 storel 7
 jump lblwhile2
 lblfimwhile2: nop
 pushl 9
 writei
 stop
```

- Ler e armazenar os elementos de um vetor de comprimento N; ordenar esse vetor por trocas diretas e imprimir os valores por ordem decrescente

  Código da nossa linguagem:

```
F Ordenacao {
    INT n;
    INT [5] vetor;
    INT i;
    INT j;
    INT min;
    INT aux;
    INT ca;

    n = 5;

    i = 0;
    WHILE( i INF n){
        [ i ] vetor = read;
        i = i + 1;
    }

    i = 0;
    ca = n - 1;
    WHILE( ca SUP i){
        min = i;
        j = i + 1;
        WHILE( n SUP j ){
            IF([ j ] vetor INF [ min ] vetor){
                min = j;
            }
            ELSE{}
            j = j + 1;
        }
        IF(i DIFFERENT min){
            aux = [ i ] vetor;
            [ i ] vetor = [ min ] vetor;
            [ min ] vetor = aux;
        }
        ELSE{}
        i = i + 1;
    }

    i = n - 1;
    WHILE(i SUPEQUAL 0){
        write [ i ] vetor;
        i = i - 1;
    }
}
```

Código máquina respectivo:

```
start
pushi 0
pushn 5
pushi 0
pushi 0
pushi 0
pushi 0
pushi 0
pushi 5
storel 0
pushi 0
storel 6
lblwhile1: nop
pushl 6
pushl 0
inf
jz lblfimwhile1
pushfp
pushi 1
padd
pushl 6
read
atoi
storen
pushl 6
```

```
pushi 1
add
storel 6
jump lblwhile1
lblfimwhile1: nop
pushi 0
storel 6
pushl 0
pushi 1
sub
storel 10
lblwhile2: nop
pushl 10
pushl 6
sup
jz lblfimwhile2
pushl 6
storel 8
pushl 6
pushi 1
add
storel 7
lblwhile3: nop
pushl 0
pushl 7

sup
jz lblfimwhile3
pushfp
pushi 1
padd
pushl 7
loadn
pushfp
pushi 1
padd
pushl 8
loadn
inf
lblif1: nop
jz lblelse1
pushl 7
storel 8
jump lblelseend1
lblifend1: nop
lblelse1: nop
lblelseend1: nop
pushl 7
pushi 1
add
storel 7

jump lblwhile3
lblfimwhile3: nop
pushl 6
pushl 8
equal
pushi 1
swap
sub
lblif2: nop
jz lblelse2
pushfp
pushi 1
padd
pushl 6
loadn
storel 9
pushfp
pushi 1
padd
pushl 6
pushfp
pushi 1
padd
pushl 8
loadn
```

```
storen
pushfp
pushi 1
padd
pushl 8
pushl 9
storen
jump lblelseend2
lblifend2: nop
lblelse2: nop
lblelseend2: nop
pushl 6
pushi 1
add
storel 6
jump lblwhile2
lblfimwhile2: nop
pushl 0
pushi 1
sub
storel 6
lblwhile4: nop
pushl 6
pushi 0
supeq
jz lblfimwhile4
pushfp
pushi 1
padd
pushl 6
loadn
writei
pushl 6
pushi 1
sub
storel 6
jump lblwhile4
lblfimwhile4: nop
stop
```

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

  Código da nossa linguagem:

```
F Ordenacao {
    INT n;
    INT [5] vetor;
    INT i;

    n = 5;

    i = 0;
    WHILE( i INF n){
        [ i ] vetor = read;
        i = i + 1;
    }

    i = n - 1;
    WHILE(i SUPEQUAL 0){
        write [ i ] vetor;
        i = i - 1;
    }
}
```

  Código máquina respectivo:

```
start
pushi 0
pushn 5
pushi 0
pushi 5
storel 0
pushi 0
storel 6
lblwhile1: nop
pushl 6
pushl 0
inf
jz lblfimwhile1
pushfp
pushi 1
padd
pushl 6
read
atoi
storen
pushl 6
pushi 1
add
storel 6
jump lblwhile1
```

```
start
pushi 0
pushn 5
pushi 0
pushi 5
storel 0
pushi 0
storel 6
lblwhile1: nop
pushl 6
pushl 0
inf
jz lblfimwhile1
pushfp
pushi 1
padd
pushl 6
read
atoi
storen
pushl 6
pushi 1
add
storel 6
jump lblwhile1
```

3.2 Exemplo de erros

- Erro 1 - Declaracão de uma variável já declarada

```
F Erro {
    INT a;
    INT a;

    a = 5;
}
```

```
start
pushi 0
Erro: Variável já definida
```

- Erro 2 - Tipo incorreto de variável

```
F Erro {
    INT a;
    INT [2] b;

    [0] a = 5;
}
```

```
start
pushi 0
pushn 2
pushi 5
Erro: Tipo incompatível
```

- Erro 3 - Atribuição de um valor lógico a um inteiro

```
F Erro {
    INT a;
    BOOL x;

    a = FALSE AND TRUE;
}
```

```
start
pushi 0
pushi 0
pushi 0
pushi 1
mul
Erro: Tipo incompatível
```

Capítulo 4

Conclusão

A realização deste projeto foi fundamental para consolidar a matéria lecionada quer nas aulas práticas quer nas aulas teóricas, uma vez que as técnicas aprendidas facilitaram a implementação do problema. Estas técnicas permitiram recolher de forma clara e eficiente as informações pretendidas. O desenho de uma linguagem de programação foi bastante enriquecedor para todos os elementos do grupo pois exigiu um aumento no domínio das funcionalidades YACC.

Dado que as exigências do projeto foram cumpridas de forma eficaz, o resultado final pode ser considerado positivo, apesar de existir sempre espaço para melhorias, tanto na implementação como no desenho da solução.

Capítulo 5

Anexos

5.1 Flex

```
 %{

%}
lixo .|\n
%%

[\+\-\*\/\;\{\}\=\(\)\[\]\%] {return yytext[0];}
(?i:F) {return F;}
(?i:INT) {return INT;}
(?i:BOOL) {return BOOL;}
(?i:TRUE) {return TRUE;}
(?i:FALSE) {return FALSE;}
(?i:EQUAL) {return EQUAL;}
(?i:DIFFERENT) {return DIFFERENT;}
(?i:SUP) {return SUP;}
(?i:INF) {return INF;}
(?i:SUPEQUAL) {return SUPEQUAL;}
(?i:INFEQUAL) {return INFEQUAL;}
(?i:AND) {return AND;}
(?i:OR) {return OR;}
(?i:WRITE) {return WRITE;}
(?i:READ) {return READ;}
(?i:IF) {return IF;}
(?i:ELSE) {return ELSE;}
(?i:WHILE) {return WHILE;}
[a-zA-z]+ {yylval.sval = strdup(yytext);  return id;}
[-]?[0-9]+ {yylval.ival = atoi(yytext); return num;}
{lixo} {;}

%%
int yywrap() {
return 1;
}
```

5.2 YACC

```
 %{
#include <stdio.h>
#include <stdlib.h>
#include <search.h>
#include <string.h>
#include "./stack.h"

typedef struct variavel{
char *nome, *tipo;
int nrLinha;
int nrColuna;
int endereco;
}*Variavel;

int yylex();
int yyerror();
void addVariavel(char *nome, char*tipo, int l, int c);
Variavel getVariavel(char* nome);
void cantBeType(char *s, char *s1);
void checkType(char *variavel, char *tipo);


int nrVariaveis = 0, nrIF = 0, nrCiclos = 0, end = 0;
char str[20];
Variavel aux;
%}

%union{ int ival; char code; char* sval;}

%token <ival>num
%token <sval>id
%token F INT BOOL WRITE READ TRUE FALSE EQUAL DIFFERENT SUP SUPEQUAL INF INFEQUAL AND OR IF ELSE WHILE
%type <sval> ID CONST Termo OpLog OpLogAO OpArit

%%

Programa:   Funcoes {;}
;
Funcoes:   Funcao
| Funcoes Funcao  {;}
;
Funcao:   F id '{'  {printf("start\n");} Declaracoes
Instrucoes '}' {printf("stop\n");}
;
Declaracoes:  Declaracao
| Declaracoes Declaracao
;
Declaracao:   INT id ';'
{addVariavel($2,"INTEGER",0,0);printf("pushi 0\n");nrVariaveis++;}
| INT '[' num ']' id ';'
{addVariavel($5,"ARRAY",$3,0);printf("pushn %d\n",$3);nrVariaveis+=$3;}
| INT '[' num ']' '[' num ']' id ';'
```

```
{addVariavel($8,"MATRIX",$3,$6);printf("pushn %d\n",$3*$6);nrVariaveis+=$3*$6;}
| BOOL id ';'
{addVariavel($2,"BOOL",0,0);printf("pushi 0\n");nrVariaveis++;}
;
Instrucoes:   /* empty */
| Instrucoes Instrucao {;}
;
Instrucao:   Atribuicao ';'{;}
| IO ';'{;}
| SE  {;}
| CICLO {;}
;
CICLO:   WHILE '('
{insertWHILE(++nrCiclos);sprintf(str,"%d",nrCiclos);printf("lblwhile%s: nop\n",str);}
OpLogAO
{printf("jz lblfimwhile%s\n",str);}
')' '{' Instrucoes
{sprintf(str, "%d", rmvWHILE());printf("jump lblwhile%s\nlblfimwhile%s: nop\n",str,str);}
'}'
;
SE:  IF '(' OpLogAO
{insertIF(++nrIF);sprintf(str, "%d", nrIF);printf("lblif%s: nop\njz lblelse%s\n",str,str);}
')' '{' Instrucoes
{sprintf(str, "%d", tellmeIF());printf("jump lblelseend%s\nlblifend%s: nop\n",str,str);}
'}' SENAO
;
SENAO: ELSE '{'
{sprintf(str, "%d", tellmeIF());printf("lblelse%s: nop\n",str);}
Instrucoes
{sprintf(str, "%d", rmvIF());printf("lblelseend%s: nop\n",str);}
'}'{;}
;
IO: WRITE Termo    {printf("writei\n");}
;
Atribuicao:   id '=' Op
{printf("storel %d\n", getVariavel($1)->endereco);}
| id '=' OpLogAO
{checkType($1,"BOOL");printf("storel %d\n", getVariavel($1)->endereco);}
| '[' num ']' id '=' Op
{checkType($4,"ARRAY");printf("storel %d\n", getVariavel($4)->endereco + $2);}
| '[' id ']' id '='
{checkType($4,"ARRAY");printf("pushfp\npushi %d\npadd\npushl %d\n", getVariavel($4)->endereco,getVariavel($2)
Op
{printf("storen\n");}
| '[' num ']' '[' num ']' id '=' Op
{checkType($7,"MATRIX");aux = getVariavel($7); printf("storel %d\n", aux->endereco + aux->nrColuna*$2 + $5);}
| '[' num ']' '[' id ']' id '='
{checkType($7,"MATRIX");printf("pushfp\npushi %d\npadd\n", getVariavel($7)->endereco);/*
end inicial matriz */printf("pushi %d\npushi %d\nmul\npushl %d\nadd\n",
getVariavel($7)->nrColuna,$2, getVariavel($5)->endereco);/* pos na matriz*/}
Op  {printf("storen\n");}
| '[' id ']' '[' num ']' id '=' {checkType($7,"MATRIX");
printf("pushfp\npushi %d\npadd\n", getVariavel($7)->endereco);/* end inicial matriz */
printf("pushi %d\npushl %d\nmul\npushi %d\nadd\n", getVariavel($7)->nrColuna,
```

```
getVariavel($2)->endereco, $5);/* pos na matriz*/}
Op  {printf("storen\n");}
| '[' id ']' '[' id ']' id '='  {checkType($7,"MATRIX");
printf("pushfp\npushi %d\npadd\n", getVariavel($7)->endereco);/* end inicial matriz */
printf("pushi %d\npushl %d\nmul\npushl %d\nadd\n", getVariavel($7)->nrColuna,
getVariavel($2)->endereco, getVariavel($5)->endereco);/* pos na matriz*/}
Op  {printf("storen\n");}
;
Op:   READ  {printf("read\natoi\n");}
| OpArit {;}
;

OpArit:       Termo {;}
| OpArit '+' Termo {printf("add\n");}
| OpArit '-' Termo {printf("sub\n");}
| OpArit '*' Termo {printf("mul\n");}
| OpArit '/' Termo {printf("div\n");}
| OpArit '%' Termo {printf("mod\n");}
;
OpLogAO : OpLog  {;}
| OpLogAO AND OpLog  {printf("mul\n");}
| OpLogAO OR OpLog  {printf("add\npushi 0\nsup\n");}
;
OpLog:   Termo  {;}
| Termo EQUAL Termo {printf("equal\n");}
| Termo DIFFERENT Termo {printf("equal\npushi 1\nswap\nsub\n");}
| Termo SUP Termo {printf("sup\n");}
| Termo INF Termo {printf("inf\n");}
| Termo SUPEQUAL Termo {printf("supeq\n");}
| Termo INFEQUAL Termo {printf("infeq\n");}
;
Termo:   CONST {$$ = $1;} // $$ INTEGER OU BOOL
| ID
{$$ = $1;} // $$ INTEGER, BOOL, ARRAY ou MATRIZZ
;
ID:   id
{$$ = getVariavel($1)->tipo;printf("pushl %d\n", getVariavel($1)->endereco);}
| '[' num ']' id
{aux = getVariavel($4); printf("pushl %d\n", aux->endereco + $2);}
| '[' id ']' id  {aux = getVariavel($4);
printf("pushfp\npushi %d\npadd\npushl %d\nloadn\n",
getVariavel($4)->endereco,getVariavel($2)->endereco);}
| '[' num ']' '[' num ']' id
{aux = getVariavel($7); printf("pushl %d\n", aux->endereco + aux->nrColuna*$2 + $5);$$ = $7;}
| '[' num ']' '[' id ']' id
{printf("pushfp\npushi %d\npadd\n", getVariavel($7)->endereco);/* end inicial matriz */

printf("pushi %d\npushi %d\nmul\npushl %d\nadd\nloadn\n",
getVariavel($7)->nrColuna,$2, getVariavel($5)->endereco);/* pos na matriz*/}
| '[' id ']' '[' num ']' id
{printf("pushfp\npushi %d\npadd\n", getVariavel($7)->endereco);/* end inicial matriz */
printf("pushi %d\npushl %d\nmul\npushi %d\nadd\nloadn\n", getVariavel($7)->nrColuna,
getVariavel($2)->endereco, $5);/* pos na matriz*/}
| '[' id ']' '[' id ']' id
```

```
{printf("pushfp\npushi %d\npadd\n", getVariavel($7)->endereco);/* end inicial matriz */
printf("pushi %d\npushl %d\nmul\npushl %d\nadd\nloadn\n", getVariavel($7)->nrColuna,
getVariavel($2)->endereco, getVariavel($5)->endereco);/* pos na matriz*/}
;
CONST:   num {printf("pushi %d\n",$1);}
| TRUE {printf("pushi 1\n");}
| FALSE {printf("pushi 0\n");}
;
%%
```

5.3 Funçoes

```
 void checkType(char *variavel, char *tipo){
Variavel v = getVariavel(variavel);
if(strcmp(v->tipo,tipo) == 0){return;}
else{ yyerror("Tipo incompatível\n"); exit(1);}
}

void cantBeType(char *s, char *s1){
if(strcmp(s,s1)==0){
yyerror("Tipo incompatível\n"); exit(1);
}
}
int yyerror(char* s) {
printf("Erro: %s\n", s);
return 1;
}

void addVariavel(char *nome, char*tipo, int l, int c){
ENTRY item, *r;

Variavel v = (Variavel)malloc(sizeof(struct variavel));
v->nome = strdup(nome);
v->tipo = strdup(tipo);
v->endereco = nrVariaveis;
v->nrLinha = l;
v->nrColuna = c;

item.key = strdup(v->nome);
item.data =  v;

if(hsearch(item,FIND) == NULL){
hsearch(item, ENTER);
}
else{
yyerror("Variável já definida\n");
exit(1);
}
}

Variavel getVariavel(char* nome){
ENTRY item, *resultado;
Variavel found_item = NULL;
```

```
item.key = strdup(nome);
resultado = hsearch(item, FIND);
if(resultado != NULL){
found_item = resultado->data;
}
return (found_item);
}


void checkType(char *variavel, char *tipo){
Variavel v = getVariavel(variavel);
if(strcmp(v->tipo,tipo) == 0){return;}
else{ yyerror("Tipo incompatível\n"); exit(1);}
}

void cantBeType(char *s, char *s1){
if(strcmp(s,s1)==0){
yyerror("Tipo incompatível\n"); exit(1);
}
}
int yyerror(char* s) {
printf("Erro: %s\n", s);
return 1;
}

void addVariavel(char *nome, char*tipo, int l, int c){
ENTRY item, *r;

Variavel v = (Variavel)malloc(sizeof(struct variavel));
v->nome = strdup(nome);
v->tipo = strdup(tipo);
v->endereco = nrVariaveis;
v->nrLinha = l;
v->nrColuna = c;

item.key = strdup(v->nome);
item.data =  v;

if(hsearch(item,FIND) == NULL){
hsearch(item, ENTER);
}
else{
yyerror("Variável já definida\n");
exit(1);
}
}

Variavel getVariavel(char* nome){
ENTRY item, *resultado;
Variavel found_item = NULL;

item.key = strdup(nome);
resultado = hsearch(item, FIND);
```

```
if(resultado != NULL){
found_item = resultado->data;
}
return (found_item);
}
```