Universidade do Minho

Licenciatura em Ciências da Computação

PLC - Trabalho Prático 2

Grupo nº17

Rui Jordão Sampaio Gonçalves
(A91652)

10 de janeiro de 2024

Conteúdo

Capítulo 1

Introdução

No âmbito da disciplina de Processamento de Linguagens e Compiladores foi proposto pelos docentes o desenvolvimento de uma Linguagem de Programação Imperativa simples e de um compilador para reconhecer programas escritas nessa linguagem gerando o respetivo código Assembly da Máquina Virtual VM.

Neste documento está apresentada a gramática e a sintaxe da nossa linguagem, bem como alguns testes com código escrito na nossa linguagem e o respetivo código Assembly gerado.

Capítulo 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* variáveis atómicas do tipo *inteiro*, com os quais se podem realizar as habituais operações aritméticas, relacionais e lógicas;
- *efetuar* instruções algorítmicas básicas como a *atribuição do valor de expressões numéricas a variáveis*;
- *ler* do *standard input* e *escrever* no *standard output*;
- *efetuar* instruções *condicionais* para controlo do fluxo de execução;
- *efetuar* instruções cíclicas para controlo do fluxo de execução, permitindo o seu aninhamento.
  Note que deve implementar pelo menos o ciclo **while-do**, **repeat-until** ou **for-do**.

Adicionalmente deve ainda suportar, à sua escolha, uma das duas funcionalidades seguintes:

- *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 (ındice inteiro);
- *definir e invocar subprogramas* sem parâmetros mas que possam retornar um resultado do tipo inteiro.

Capítulo 3

Concepção da Solução

Neste capítulo vamos apresentar:

- A sintaxe da minha linguagem
- Os símbolos
- O desenho da gramática independente de contexto

Nota que: não foi possível implementar nenhuma das funcionalidades extras.

3.1 Sintaxe da Linguagem

A sintaxe da linguagem é a seguinte

3.1.1 Declaração de variáveis

```
1         int x;
2
```

3.1.2 Atribuição de variáveis

```
1         x = 10;
2
```

3.1.3 Operadores de comparação

```
1         x <= y
2         x >= y
3         x < y
4         x > y
5         x == y
6         x != y
7
```

3.1.4 Operações numéricas

```
        x + y
        x - y
        x / y
        x * y

```

3.1.5 Operadores lógicos

```
        x E y
        x OU y
        Negar x

```

3.1.6 Instruções condicionais

```
        Se (Condicao) Faz Instrucoes ;
        Se (Condicao) Faz instrucoes Senao Instrucoes ;

```

3.1.7 Ciclo while

```
        Enquanto (Condicao) Faz Instrucoes ;

```

3.1.8 Input/Output

```
_________Escrever (String) ;
_________Escrever (20) ;
_________Escrever (a) ;
_________Ler () ;
_________Notar que: A funcionalidade de leitura n o est
    finalizada...

```

3.2 Símbolos

Os simbolos da linguagem são os seguintes:

```
Inicio: r'Inicio'
Fim: r'Fim'
Num: r'[+\-]?\d+'
PontoVirgula: r'\;'
ParRetoEsq: r'\['
```

```
    ParRetoDir: r'\]'
    Soma: r'\+'
    Subtracao: r'\-'
    Multiplicacao: r'\*'
    Divisao: r'\/'
    ParCurvoEsq: r'\('
    ParCurvoDir: r'\)'
    Ler: r'Ler'
    Escrever: r'Escrever'
    Senao: r'Senao'
    Se: r'Se'
    Faz: r'Faz'
    Enquanto: r'Enquanto'
    Para: r'Para'
    E: r'E'
    OU: r'OU'
    Negar: r'Negar'
    Diferente: r'\!\='
    Igual: r'\=\='
    Vale: r'\='
    MaiorIgual: r'\>\='
    MenorIgual: r'\<\='
    Maior: r'\>'
    Menor: r'\<'
    String: r'"([^"]|(\\n))*"'
    Var: r'[a-zA-Z]\w*'
```

3.3 Desenho da GIC

A minha linguagem é gerada pela seguinte grámatica independente de contexto:

```
Programa : Inicio Corpo Fim
    Corpo : Declaracoes Instrucoes
    Declaracoes : Declaracoes Declaracao
        |
    Declaracao : Tipo Var PontoVirgula
        | Tipo Var ParRetoEsq Expressao ParRetoDir PontoVirgula
        | Tipo Var ParRetoEsq Expressao ParRetoDir ParRetoEsq Expressao ParRetoDir P
    Instrucoes : Instrucoes Instrucao
        |
    Instrucao : Atribuicao
        | Leitura
```

```
    | Escrita
    | Selecao
    | Repeticao
Atribuicao : Var Vale Expressao PontoVirgula
           | Var ParRetoEsq Expressao ParRetoDir Vale Expressao PontoVirgula
           | Var ParRetoEsq Expressao ParRetoDir ParRetoEsq Expressao ParRetoDi
Condicao : Expressao
    | Expressao Igual Expressao
    | Expressao Diferente Expressao
    | Expressao MenorIgual Expressao
    | Expressao MaiorIgual Expressao
    | Expressao Menor Expressao
    | Expressao Maior Expressao
Expressao : Termo
    | Expressao Soma Termo
    | Expressao Subtracao Termo
    | Expressao OU Termo
Termo : Fator
    | Termo Multiplicacao Fator
    | Termo Divisao Fator
    | Termo E Fator
Fator : Frase
    | Var ParRetoEsq Expressao ParRetoDir
    | Var ParRetoEsq Expressao ParRetoDir ParRetoEsq Expressao ParRetoDir
    | ParCurvoEsq Condicao ParCurvoDir
    | Negar Fator
Leitura : Ler ParCurvoEsq Expressao ParCurvoDir PontoVirgula
Escrita : Escrever ParCurvoEsq Expressao ParCurvoDir PontoVirgula
Selecao : Se ParCurvoEsq Condicao ParCurvoDir Faz Instrucoes PontoVirgula
    | Se ParCurvoEsq Condicao ParCurvoDir Faz Instrucoes Senao Instrucoes Ponto
Repeticao : Enquanto ParCurvoEsq Condicao ParCurvoDir Faz Instrucoes PontoVirgul
    | Para ParCurvoEsq Atribuicao PontoVirgula Condicao PontoVirgula Atribuicao
Frase : String
    | Lista_Palavras
Lista_Palavras : Palavra
            | Lista_Palavras Palavra
Palavra : Var
    | Num
"""
```

Capítulo 4

Exemplos de funcionamento

4.0.1 (Se Simples

```
Inicio

int a;
int b;

a = 5;
b = 10;

Se (a > b) Faz
    Escrever (" a maior que b ");
Senao
    Escrever (" b maior que a ");
;

Fim

```

Código Assembly gerado:

```
PUSHI 0
PUSHI 0
START
PUSHI 5
STOREG 0
PUSHI 10
STOREG 1
PUSHG 0
PUSHG 1
SUP
JZ IF0
PUSHS " a maior que b "
WRITES

JUMP ELSE0
IF0:
```

```
PUSHS " b maior que a "
WRITES

ELSE0:
STOP

```

4.0.2 Se aninhado

```
Inicio

int a;
int b;
int c;

a = 30;
b = 20;
c = 15;

Se (a > b) Faz
    Se (c > a) Faz
        Escrever (15);
    Senao
        Escrever (30);
    ;
Senao
    Escrever(20);
;

Fim

```

Código Assembly gerado:

```
PUSHI 0
PUSHI 0
PUSHI 0
START
PUSHI 30
STOREG 0
PUSHI 20
STOREG 1
PUSHI 15
STOREG 2
PUSHG 0
PUSHG 1
SUP
JZ IF1
PUSHG 2
PUSHG 0
SUP
JZ IF0
```

```
PUSHI 15
WRITEI
JUMP ELSE0
IF0:
PUSHI 30
WRITEI
ELSE0:
JUMP ELSE1
IF1:
PUSHI 20
WRITEI
ELSE1:
STOP

```

4.0.3 Enquanto simples

```
Inicio

int a;
int N;

a = 0;
N = 5;

Enquanto (a < N) Faz
    a = a + 1;
;

Escrever("a: ");
Escrever(a);

Fim

```

Código Assembly gerado:

```
PUSHI 0
PUSHI 0
START
PUSHI 0
STOREG 0
PUSHI 5
STOREG 1
WHILE0:
PUSHG 0
PUSHG 1
INF
JZ ENDWHILE0
PUSHG 0
 PUSHI 1
ADD
```

```
16
17 STOREG 0
18 JUMP WHILE0
19 ENDWHILE0:
20 PUSHS "a: "
21 WRITES
22
23 PUSHG 0
24 WRITEI
25 STOP
26
```

4.0.4 Enquanto aninhado

```
1
2 Inicio
3
4 int a;
5 int N;
6 int b;
7
8 a = 0;
9 b = 0;
10 N = 2;
11
12 Enquanto (a < 2) Faz
13     Enquanto (b < 3) Faz
14         b = b + 1;
15     ;
16     a = a + 1;
17 ;
18
19 Escrever("a: ");
20 Escrever(a);
21
22 Escrever(", ");
23
24 Escrever("b: ");
25 Escrever(b);
26
27 Fim
28
```

Código Assembly gerado:

```
1 PUSHI 0
2 PUSHI 0
3 PUSHI 0
4 START
5 PUSHI 0
6 STOREG 0
7 PUSHI 0
```

```
STOREG 2
PUSHI 2
STOREG 1
WHILE1:
PUSHG 0
PUSHI 2
INF
JZ ENDWHILE1
WHILE0:
PUSHG 2
PUSHI 3
INF
JZ ENDWHILE0
PUSHG 2
 PUSHI 1
ADD

STOREG 2
JUMP WHILE0
ENDWHILE0:
PUSHG 0
 PUSHI 1
ADD

STOREG 0
JUMP WHILE1
ENDWHILE1:
PUSHS "a: "
WRITES

PUSHG 0
WRITEI
PUSHS ", "
WRITES

PUSHS "b: "
WRITES

PUSHG 2
WRITEI
STOP

```

4.0.5 Negar

```
Inicio

    int a;
    int b;

    a = 0;
    b = 1;

```

```
    Se (Negar a) Faz
        Escrever (a);
    Senao
        Escrever (b);
    ;
Fim

```

Código Assembly gerado:

```
PUSHI 0
PUSHI 0
START
PUSHI 0
STOREG 0
PUSHI 1
STOREG 1
PUSHG 0
NOT
JZ IF0
PUSHG 0
WRITEI
JUMP ELSE0
IF0:
PUSHG 1
WRITEI
ELSE0:
STOP

```

4.0.6 E

```
Inicio

    int a;
    int b;

    a = 0;
    b = 1;

    Se (a E b) Faz
        Escrever (a);
    Senao
        Escrever (b);
    ;
Fim

```

Código Assembly gerado:

```
PUSHI 0
PUSHI 0
START
PUSHI 0
STOREG 0
PUSHI 1
STOREG 1
PUSHG 0
PUSHG 1
AND
JZ IF0
PUSHG 0
WRITEI
JUMP ELSE0
IF0:
PUSHG 1
WRITEI
ELSE0:
STOP

```

4.0.7 SOMA

```
Inicio

    int a;
    int b;
    int c;
    int d;

    a = 20;
    b = 30;
    c = 40;
    d = a + b + c;
    Escrever ("d: ");
    Escrever (d);

Fim

```

Código Assembly gerado:

```
PUSHI 0
PUSHI 0
PUSHI 0
PUSHI 0
START
PUSHI 20
STOREG 0
PUSHI 30
STOREG 1
PUSHI 40
```

```
STOREG 2
PUSHG 0
 PUSHG 1
ADD

 PUSHG 2
ADD

STOREG 3
PUSHS "d: "
WRITES

PUSHG 3
WRITEI
STOP

```

Capítulo 5

Conclusão

Fazendo uma retrospetiva referente a este trabalho prático, infelizmente não foi possível cumprir todos os objetivos propostos.

No entanto a realização deste trabalho prático fez com que fosse introduzido os conceitos do funcionamento do módulo Lexer e do módulo Yacc, nomeadamente como funciona o reconhecimento de tokens e a implementação da nossa gramática. A geração de código Assembly foi sem dúvida também um ponto positivo deste trabalho pois permitiu entender melhor a linguagem e as suas instruções.

Aproveito para enumerar funcionalidades que seriam interessantes introduzir numa próxima atualização deste programa: ciclo for, arrays, alargar os tipo de dados (por exemplo float ou double), fazer a leitura do terminal e guardar o valor numa variável previamente definida, introduzir precedência das operações aritméticas, introduzir comentários, melhorar a uniformidade dos outputs gerados pelo meu programa (alteração apenas estética).