Processamento de Linguagens e Compiladores
(3º ano de LCC)

Galo

TP3

Grupo 14

Artur Queiroz	Rafael Fernandes	Rafaela Pinho
A77136	A78242	A77293

15 de Janeiro de 2018

Resumo

Neste relatório apresentamos a linguagem que criamos e o complidor que gera o código para a Máquina Virtual VM.

Conteúdo

Capítulo 1

Introdução

Neste relatório apresentamos o último trabalho da unidade curricular "Processamento de Linguagens e Compliadores". Este consiste em desenvolver um processador de linguagens usando o método da tradução dirigida pela sintaxe. Também devemos desenvolver um compilador que gera o código para uma máquina de stack virtual. Utilizaremos a ferramenta Yacc para gerar compiladores baseados em gramáticas tradutoras.

Como todos os outros trabalhos, este também tem como objetivo aumentar a experência no uso do ambiente Linux, da linguagem C e ferramentas de apoio à programção.

Este relatório está dividido em 4 partes. No primeiro capítulo encontramos a parte introdutório a este trabalho, onde explicamos no que consiste. No capítulo 2 (dois) apresentamos os requisitos e a linguagem que criamos. No terceiro mostramos as decisões e alguns teste efectuados. No último temos a conclusão e os objetivos do trabalho futuro.

Capítulo 2

Galo e Compilador

2.1 Descrição informal do problema

Neste trabalho foi pedido para criarmos uma linguagem de programação imperativa e desenvolver um compilador para a linguagem criada.

Na linguagem as declarações de variáveis devem ser colocadas no início do programa, não pode haver re-declarações e não se pode usar variáveis sem estar declaradas primeiro. Caso não seja atribuido uma valor à variável depois da declaração, esta ficará com o valor zero.

O complilador deve gerar o código assembly para a Máquina Virtual VM.

2.2 Especificação dos requisitos

Para este tarbalho a linguagem que criamos tem de conter os seguintes requisitos:

1. Declarar e manusear variáveis atómicas do tipo inteiro e estruturas do tipo array de inteiros.

2. Ler do standard input e escrever no standard output.

3. Fazer instruções básicas como a atribuição de expressões a variáveis.

4. definir e invocar subprogramas sem parâmetros mas que possam retornar um resultado atómico.

5. efetuar instruções para controlo do fluxo de execução—condicional e cíclica—que possam ser aninhadas.

Também devrá conter um conjunto de testes (escritos na nossa linguagem) que tem de ter no mínino os 6 exemplos seguintes:

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

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

3. Ler N (constante do programa) números e calcular e imprimir o seu produtório.

4. Contar e imprimir os números impares de uma sequência de números naturais.

5. 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.

6. Ler e armazenar N números num array; imprimir os valores por ordem inversa.

2.3 Expressões regulares

As expressões regulares usadas foram:

1)

2.4 A nossa linguagem

2.4.1 Galo

Como já referido em cima, foi-nos pedidos para criar uma linguagem de programação. Decidimos chamar de Galo por ser um símbolo típico de Potugal, e atribuimos .gl para a extensão.

Para a definirmos utilizamos uma gramática independente do contexto, em que tomamos certas decisões que serão especificadas.

O Galo reconhece os segintes tipos: números inteiros (int), números décimais (float) e conjunto de caractéres (string). A linguagem usa os habituais síbolos de comparação, como <=, >=, == e ! =. Utiliza o "e"e o "ou"como símbolos de operadores lógicos.

Como a nossa linguagem é muito parecida ao C, para fazer o "ite"utilizamos o "se"e o "senao"(tanto minúsculo como maiúsculo) para o "while"usamos o

"enq"ou "ENQ".

Existe as funções "ler?()"e "escrever?()"que são, respetivamente, a função de leitura no teclado e de escrita no ecrã. (? = i ou s, i-inteiro, s-string)

A nossa linguagem está definida pela seguinte GIC:

```
1 ProgG: ProgG Se
2      | ProgG Enq
3      | ProgG Atrib ';'
4      | ProgG VAR '=' Expr ';'
5      | ProgG VAR '[' NUM ']' '=' Expr ';'
6      | ProgG CriaFun
7      | ProgG Funcao ';'
8      | ProgG ';'
9      | ProgG COM
10      | %empty

11 ProgF: ProgF Se
12      | ProgF Enq
13      | ProgF Atrib ';'
14      | ProgF VAR '=' Expr ';'
15      | ProgF VAR '[' NUM ']' '=' Expr ';'
16      | ProgF Funcao ';'
17      | ProgF ';'
18      | ProgF COM
19      | ProgF RETURN Expr ';'
20      | %empty

21 Prog: Prog Se
22     | Prog Enq
23     | Prog Atrib ';'
24     | Prog VAR '=' Expr ';'
25     | Prog VAR '[' NUM ']' '=' Expr ';'
26     | Prog Funcao ';'
27     | Prog ';'
28     | Prog COM
29     | %empty

30 Funcao: VAR Lexpr

31 CriaFun: TIPO VAR '('
32        | Ltipo '{' ProgF '}'

33 Atrib: TIPO VAR
34      | TIPO VAR '[' NUM ']'
35      | Igual

36 Igual: TIPO VAR '='
37      | Igual Expr

38 Lexpr: '(' ')'
39      | '(' Eexpr ')'

40 Eexpr: Expr
41      | Eexpr ',' Expr
```

```
42 Ltipo: ')'
43      | Etipo ')'

44 Etipo: TIPO VAR
45      | Etipo ',' TIPO VAR

46 Se: SE Cond
47   | Se '{' Prog '}' CASO Cond
48   | Se '{' Prog '}' SENAO
49   | Se '{' Prog '}'

50 Enq: ENQ
51    | Enq Cond
52    | Enq '{' Prog '}'

53 Cond: NUM
54     | '(' Expr EQ Expr ')'
55     | '(' Expr NEQ Expr ')'
56     | '(' Expr '<' Expr ')'
57     | '(' Expr '>' Expr ')'
58     | '(' Expr LEQ Expr ')'
59     | '(' Expr GEQ Expr ')'
60     | '(' Cond E Cond ')'
61     | '(' Cond OU Cond ')'
62     | '!' Cond

63 Sexpr: VAR
64      | NUM
65      | FLOAT
66      | VAR '[' Expr ']'
67      | Funcao
68      | STR

69 Expr: '(' Expr '+' Expr ')'
70     | '(' Expr '-' Expr ')'
71     | '(' Expr '*' Expr ')'
72     | '(' Expr '/' Expr ')'
73     | '(' Expr '%' Expr ')'
74     | '(' Expr ')'
75     | Sexpr
```

Capítulo 3

Codificação e Testes

3.1 Problemas de implementação, Decisões e Alternativas

3.1.1 Problemas de implementação

3.1.2 Decisões

1) O "e"(&&) está definida pela multuplicação e o "ou"(||) pela adição.

Tabela 3.1: Tabela do E

*(E)	0	1
0	0	0
1	0	1

Tabela 3.2: Tabela do OU

+(OU)	0	1
0	0	1
1	1	2

2) Não se pode declarar mais do que uma variável numa linha, ou seja todas as declarações são individuais.
   Exemplo:
   int a = 2 , c = 0;
   terá de ser:
   int a = 2;
   int c = 0;

3) Não se pode fazer "return"dentro dos Se's e dos Enq's.

4) Nas expressões numéricas, as operações binárias têm de estar sempre dentro de parênteses.
   Exemplo:
   int a = (1+(2*3))

5) todo o codigo dentro de se's, enq's e funções começa em { e acaba em }.

3.1.3 Alternativas

3.2 Testes realizados e Resultados

3.2.1 Exemplos escritos na nossa linguagem

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

```
int a = leri();
int b = leri();
int c = leri();
int d = leri();

se ((a==b) e ((b==c) e (c==d))) {
        escrevers("É um quadrado\n");
}
senao {
        escrevers("Não é um quadrado\n");
}
```

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

```
escrevers("Escreva o número de elementos do array:\n");
int N = leri();

int i = 0;
int a = 0;
int res = 0;

se (N!=0){
        res = leri();
        i = 1;
        enq (i < N){
                a = leri();

                se (res > a){
                        res = a;
                }

                i = (i+1);
        }


        escrevers("O menor número foi o ");
        escreveri(res);
```

```
        escrevers("\n");

}
senao{
        escrevers("Não leu nenhum número\n");
}
```

3. Ler N (constante do programa) números e calcular e imprimir o seu produtório.

```
escrevers("Vão se ler 5 números\n");

int N = 5;
int i = 0;
int r = 1;

enq(i < N){

        r = (r * leri());

        i = (i + 1);
}

escrevers("O produtório desta sequencia de 5 números é ");
escreveri(r);
escrevers("\n");
```

4. Contar e imprimir os números impares de uma sequência de números naturais.

```
escrevers("Digitar uma sequência de números, termina quando for zero\n");

int cont = 0;
int i = leri();

enq(i != 0){
        se ((i % 2) == 1){
                cont = (cont + 1);
                escreveri(i);
                escrevers(" \n");
        }
        i = leri();
}

escrevers("Foram lidos ");
escreveri(cont);
escrevers(" impares \n");
```

5. 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.

```
int troca(int v, int i, int j){
        int k = v[i];
        v[i] = v[j];
        v[j] = k;
        return 0;
}

int ordena(int* v, int N){
        int i = 0; #inicio
        int j = 0; #procura
        int m ;    #pos do menor

        enq (i<(N-1)){
                j = (i + 1);
                m = i;
                enq (j<N){
                        se(v[j]>v[m]){
                                m = j;
                        }
                        j = (j + 1);
                }
                troca(v,i,m);
                i = (i + 1);
        }
        return 0;
}

int N = leri();
int i = 0;
int v[N];

enq(i<N){
        v[i] = leri();
        i = (i+1);
}

ordena(v, N);

i = 0;

enq(i<N){
        escreveri(v[i]);
        escrevers("\n");
}
```

6. Ler e armazenar N números num array; imprimir os valores por ordem inversa.

```
int N = leri();
int i = 0;
int a = 0;
int v[N];

enq (i < N){
        v[i] = leri();
        i = (i + 1);
```

```
}

enq(i > 0){
        a = v[(i-1)];
        escreveri(a);
        escrevers("\n");
        i = (i-1);
}
```

Capítulo 4

Conclusão