Not Really C - Relatório TP2

Grupo 101

Rafael Costa A102526
João Sousa A102462
João Matos A102528

1. Introdução

No âmbito da unidade curricular de Processamento de Linguagens e Compiladores, foi-nos proposto o desenvolvimento de uma linguagem de programação imperativa simples e de um compilador que gera código Assembly para a Máquina Virtual (VM). O projeto visava reforçar a nossa compreensão de engenharia de linguagens e programação generativa através do desenvolvimento de gramáticas e construção de compiladores.

A nossa linguagem, denominada "Not Really C" (NRC), foi concebida para ser uma versão simplificada do C, mantendo uma sintaxe familiar para as estruturas básicas de programação. A linguagem suporta conceitos fundamentais de programação, sendo simultaneamente suficientemente simples para fins educativos.

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 de seleção para controlo do fluxo de execução.
- efetuar instruções de repetição (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 (índice inteiro).
- definir e invocar subprogramas sem parâmetros mas que possam retornar um resultado do tipo inteiro.

Como é da praxe neste tipo de linguagens, as variáveis deverão ser declaradas no início do programa e não pode haver
re-declarações, 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 aos módulos
Yacc/ Lex do PLY/Python.
O compilador deve gerar pseudo-código, Assembly da Máquina Virtual VM.
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.

3. Concepção da Solução

a. Sintaxe da Linguagem

i. Declaração de Variáveis

```
int x;
int x = 10;
int arr[10];
int matriz[5][2];
```

ii. Operadores de Comparação

- Menor que: `<`
- Maior que: `>`
- Menor ou igual: `<=`
- Maior ou igual: `>=`
- Igual a: `==`
- Diferente de: `!=`

iii. Operações Aritméticas

- Soma: `+`
- Subtração: `-`
- Multiplicação: `*`
- Divisão: `/`
- Resto da divisão: `%`

iv. Operadores Lógicos

- E: `&&`
- OU: `||`
- NÃO: `!`

v. Estruturas de Controlo

```
if (condição) {
    // código
}

if (condição) {
    // código
} else {
    // código
}

while (condição) {
    // código
}
```

vi. Input/Output

```
read(variável);
print(expressão);
```

b. Desenho da Gramática

A gramática da linguagem segue uma estrutura semelhante ao C, com algumas simplificações. A estrutura principal do programa requer uma declaração da função main:

```
program        : "int" "main" "(" ")" "{" declarations statements "}"

declarations   : declaration declarations
               | ε

declaration    : "int" identifier array_decl ";"
               | "int" identifier "=" expression ";"
               | "int" identifier ";"

array_decl     : "[" NUMBER "]"
               | "[" NUMBER "]" "[" NUMBER "]"

statements     : statement statements
               | ε

statement      : assignment ";"
               | if_statement
               | while_statement
               | io_statement ";"

assignment     : variable "=" expression
```

```
if_statement    : "if" "(" condition ")" "{" statements "}"
                | "if" "(" condition ")" "{" statements "}" "else" "{" statements "}"

while_statement : "while" "(" condition ")" "{" statements "}"

io_statement    : "read" "(" variable ")"
                | "print" "(" expression ")"

expression      : term
                | expression "+" term
                | expression "-" term

term            : factor
                | term "*" factor
                | term "/" factor
                | term "%" factor

factor          : NUMBER
                | variable
                | "(" expression ")"

variable        : identifier
                | identifier "[" expression "]"
                | identifier "[" expression "]" "[" expression "]"

condition       : expression rel_op expression
                | condition "&&" condition
                | condition "||" condition
                | "!" condition
                | "(" condition ")"

rel_op          : "<" | ">" | "<=" | ">=" | "==" | "!="

identifier      : LETTER (LETTER | DIGIT)*

NUMBER     : DIGIT+
LETTER     : [a-zA-Z]
DIGIT      : [0-9]
```

Componentes principais da gramática:
- Declarações de variáveis no início do programa
- Suporte para expressões com precedência correta de operadores
- Acesso a arrays com uma ou duas dimensões
- Estruturas de controlo
- Operações de Input/Output

c. Funcionalidades Adicionais

i. Gestão da Tabela de Símbolos

O compilador mantém uma tabela de símbolos para controlar:
- Declarações de variáveis
- Dimensões dos arrays
- Deslocamentos de memória para variáveis

ii. Tratamento de Erros

O compilador inclui verificação de erros para:

- Variáveis não definidas
- Redeclaração de variáveis
- Limites dos arrays
- Erros de sintaxe

4. Exemplos

a. Operações com Arrays

```
int main() {
    int x = 5;
    int p = 30;
    int arr [10];
    arr [2] = x + p;
    if (x > 0) {
        print(arr[2]);
    }
}
```

```
PUSHI 5
STOREG 0
PUSHI 30
STOREG 1
PUSHI 0
STOREG 2
PUSHI 0
STOREG 3
PUSHI 0
STOREG 4
PUSHI 0
STOREG 5
PUSHI 0
STOREG 6
PUSHI 0
STOREG 7
PUSHI 0
```

```
STOREG 8
PUSHI 0
STOREG 9
PUSHI 0
STOREG 10
PUSHI 0
STOREG 11
PUSHGP
PUSHI 2
PUSHI 2
ADD
PUSHG 0
PUSHG 1
ADD
STOREN
PUSHG 0
PUSHI 0
SUP
JZ L2
PUSHGP
PUSHI 2
PUSHI 2
ADD
LOADN
WRITEI
WRITELN
L2: NOP
```

b. Operações com Matrizes

```
int main(){
    int x = 12;
    int p = 5;
    int mat[5][2];
    mat[2][0] = x*p;
    print(mat[2][0]);
}
```

```
PUSHI 12
STOREG 0
PUSHI 5
STOREG 1
PUSHI 0
STOREG 2
PUSHI 0
STOREG 3
PUSHI 0
STOREG 4
```

```
PUSHI 0
STOREG 5
PUSHI 0
STOREG 6
PUSHI 0
STOREG 7
PUSHI 0
STOREG 8
PUSHI 0
STOREG 9
PUSHI 0
STOREG 10
PUSHI 0
STOREG 11
PUSHGP
PUSHI 2
PUSHI 2
PUSHI 2
MUL
PUSHI 0
ADD
ADD
PUSHG 0
PUSHG 1
MUL
STOREN
PUSHGP
PUSHI 2
PUSHI 2
PUSHI 2
MUL
PUSHI 0
ADD
ADD
LOADN
WRITEI
WRITELN
```

c. Média Aritmética

```
int main() {
    int a = 10;
    int b = 20;
    int media;

    media = (a + b) / 2;
    print(media);
}
```

```
PUSHI 10
STOREG 0
PUSHI 20
STOREG 1
PUSHI 0
STOREG 2
PUSHG 0
PUSHG 1
ADD
PUSHI 2
DIV
STOREG 2
PUSHG 2
WRITEI
WRITELN
```

d. Números Pares

```
int main() {
    int arr[10];
    int i = 0;

    while(i < 10) {
        arr[i] = i;
        if((arr[i] % 2) == 0){
            print(arr[i]);
        }
        i = i + 1;
    }
}
```

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

```
PUSHI 0
STOREG 8
PUSHI 0
STOREG 9
PUSHI 0
STOREG 10
L1: NOP
PUSHG 10
PUSHI 10
INF
JZ L2
PUSHGP
PUSHI 0
PUSHG 10
ADD
PUSHG 10
STOREN
PUSHGP
PUSHI 0
PUSHG 10
ADD
LOADN
PUSHI 2
MOD
PUSHI 0
EQUAL
JZ L4
PUSHGP
PUSHI 0
PUSHG 10
ADD
LOADN
WRITEI
WRITELN
L4: NOP
PUSHG 10
PUSHI 1
ADD
STOREG 10
JUMP L1
L2: NOP
```

5. Conclusão

O desenvolvimento do ‘Not Really C’ cumpriu com sucesso todos os requisitos do projeto. A linguagem oferece um ambiente de programação simples mas funcional, com suporte para estruturas básicas de programação e arrays bidimensionais. O compilador gera código assembly eficiente para a VM e inclui um tratamento adequado de erros.

Principais conquistas:
- Implementação de um pipeline completo de compilação
- Suporte para arrays unidimensionais e bidimensionais
- Suporte para loops While
- Tratamento de erros
- Sintaxe limpa e consistente

O projeto proporcionou uma experiência valiosa na concepção e implementação de compiladores, particularmente na compreensão da relação entre estruturas de linguagem de alto nível e código assembly de baixo nível.