E-fólio B: máximo 4 valores

No e-fólio B vamos considerar a geração e otimização de código intermédio, bem como os melhoramentos feitos nas análises léxica e sintática (que se refletirão nas notas do e-fólio A) da linguagem MID. Relembrando as suas características.

"a) A linguagem admite dois tipos: double e int. O tipo double pode ser representado no formato com casas decimais ou notação científica. O tipo int pode ser representado por um número em notação decimal.
Ex.: double 1.3, -1.3, 1.3e-2, -1.3e2; int: 17, -5, 0.
b) Existem apenas 3 tipos de instruções, todas terminando com o caráter ; (ponto e vírgula):
- instrução de ciclo: while(Condição) { Instruções } ;
- instrução condicional if-then-else: if(Condição) { Instruções } { Instruções } ;
- instrução de atribuição: Var = Expressão ;
Os nomes das variáveis seguem as regras da linguagem C, a expressão admite os operadores +. -, * e /, a condição admite os operadores >,<,>=,<=,==,!= (com o significado que têm na linguagem C) e dentro das chavetas pode haver 0 ou mais instruções. As chavetas no while e no if-then-else são sempre obrigatórias. Tanto nas expressões como nas condições os números são tratados como sendo do tipo double, no momento da atribuição a um inteiro é cortada a parte decimal.

Um programa na linguagem MID, tem a declaração de todas as variáveis no início (sem inicialização, é-lhes atribuído o valor 0 automaticamente).
Depois, vêm as instruções."

Pegando no exemplo do e-fólio A, vamos ver qual o código a ser gerado em TAC (Three Address Code), com o código MID em comentários:

```
// int zuvi, zeva, novi;

// double x, y, z;

// zuvi = 1;

_t0 = 1

zuvi = (int) _t0

// zeva = 2;

_t1 = 2

zeva = (int) _t1

novi = zuvi + zeva * 4 / zuvi - zeva ;

_t2 = zeva * 4
```

```
_t3 = _t2 / zuvi

_t4 = zuvi + _t3

_t5 = _t4 - zeva

novi = (int) _t5

// x = novi / 2 ;

_t6 = novi / 2

x = _t6

// while (x>y) { z = z + 1; x = x - 1; } ;

L2:

ifz x>y goto L1

_t7 = z + 1

z = _t7

_t8 = x - 1

x = _t8

goto L2

L1:

// if(z>x) { x=z; } { z=x; } ;

ifz z>x goto L3

_t9 = z

x = _t9

goto L4

L3:

_t10 = x

z = _t10

L4:

// if(y>x) { y = 0; } { } ;
```

```
ifz y > x goto L5
_t11 = 0
y = _t11
goto L6:
L5:
L6:
```

Após otimização, deverá ficar assim:

```
zuvi = 1
zeva = 2
novi = 7
x = 3.5
L2:
ifz x>y goto L1
z = z + 1
x = x - 1
goto L2
L1:
ifz z>x goto L3
x = z
goto L4
L3:
z = x
L4:
ifz y > x goto L5
y = 0
```

L5:

NOTAS IMPORTANTES:

1. Qualquer dúvida no enunciado do e-fólio deve ser esclarecida no fórum respetivo.

2. Qualquer tentativa de plágio levará à anulação do e-fólio e à atribuição da classificação 0 (zero).

3. O trabalho deve ser enviado num único ficheiro ZIP, contendo os respetivos códigos, um ficheiro readme.txt a explicar como compilar e executar o programa e um relatório de não mais do que 4 páginas a explicar a descrever o trabalho e a justificar as opções tomadas. Cotações: geração de código TAC (40%), otimização de código TAC (40%), relatório e ficheiro readme.txt (20%).