Compiladores, 2019/2020
Trabalho prático, parte 4

– Geração de Código –

Fernando Lobo

1 Introdução

Neste quarto e último trabalho iremos completar o compilador com a alteração do gerador de código. Como ponto de partida, deverá fazer download do código completo do compilador de Triangle para código TAM, disponível em:

- http://www.dcs.gla.ac.uk/~daw/books/PLPJ/

Visite o endereço web mencionado acima. Clique em 'Software' na barra do lado esquerdo e faça download do ficheiro `Triangle-tools-2.1.zip`.

O código completo do compilador está na pasta `tools-2.1`. Deverá incluir neste código as alterações que fez para as partes 1, 2 e 3, dos trabalhos práticos anteriores. Os ficheiros novos relativamente à parte 4 são os ficheiros que estão na pasta `TAM`, na pasta `Triangle/CodeGenerator` e também o ficheiro `Triangle/Compiler.java`. A classe `Compiler` implementada em `Triangle/Compiler.java` contém a função `main()`.

O trabalho consiste em modificar o gerador de código de forma a contemplar as extensões à linguagem introduzidas nas partes anteriores do trabalho com o comando `for` e o comando `case`.

Este trabalho não será submetido ao Mooshak. Deverá testar o programa criando programas de teste em Triangle, compilando-os, correndo o disassembler para inspecionar o código gerado, e correndo o respectivo código no interpretador de TAM para verificar se o programa faz aquilo que é suposto fazer.

2 Leitura e estudo prévio

Para a realização deste trabalho é muito importante a leitura do capítulo 6 e 7 do livro *Programming Language Processors in Java*, bem como o apêndice C do mesmo

livro. (Também é muito útil estudarem o código do interpretador TAM que está em `tools-2.1/TAM`.)

3 Recomendação

Sugere-se que faça o seguinte:

1. Pegue no código fonte dos autores do livro que está em `tools-2.1/Triangle`
2. Copie o conteúdo das pastas que alterou nos trabalhos 1, 2, 3 (`AbstactSyntaxTrees`, `ContextualAnalyzer`, `SyntaticAnalyzer`) para dentro de `tools-2.1/Triangle`, mas mantenha a versão original dos ficheiros,
   - `SyntaticAnalyzer/SourceFile.java`
   - `AbstractSyntaxTrees/AST.java`
3. Para fazer o trabalho propriamente dito, terá de acrescentar alguns métodos na classe `Encoder` que se encontra em `CodeGeneration/Encoder.java`, nomeadamente aqueles que vão implementar a geração de código dos comandos `for` e `case`.

Implemente primeiro a geração de código para o comando `for`, que é bem mais simples. Após certificar-se que está correctamente implementado, dedique-se à geração de código para o comando `case`, que é mais complexo.

4 Semântica dos comandos for e case

4.1 for

Recorde que a sintaxe do comando `for` tem duas variantes, podendo ser,

```
for V from E1 to E2 do C
```

ou,

```
for V from E1 downto E2 do C
```

em que `V` é um *Vname* (uma variável) do tipo Integer, `E1` e `E2` são *Expression*'s do tipo Integer, e `C` é um *SingleCommand* da linguagem Triangle.

A semântica do ciclo `for V from E1 to E2 do C` é equivalente a:

```
V := E1;
while V <= E2 do
begin
    C;
    V := V + 1
 end
```

A título de exemplo, o seguinte código em Triangle, ao ser compilado e executado, deveria escrever no ecrã: `1 2 3 4 5 6 7 8 9 10`

```
let
   var i: Integer
in
   for i from 1 to 10 do
   begin
       putint(i);
       put(' ')
   end
```

No que respeita à variante `for V from E1 downto E2 do C`, a semântica é equivalente a:

```
V := E1;
while V >= E2 do
begin
    C;
    V := V - 1
end
```

Como exemplo, o seguinte código em Triangle, ao ser compilado e executado, deveria escrever no ecrã: `10 9 8 7 6 5 4 3 2 1`

```
let
   var i: Integer
in
   for i from 10 downto 1 do
   begin
       putint(i);
       put(' ')
   end
```

4.2 case

Recorde a sintaxe do comando `case`.

```
case E of
   IL1: C1;
   IL2: C2;
   ...
   else: C0
```

em que `E` é uma *Expression* do tipo Integer, `IL1`, `IL2`, ..., são *IntegerLiteral*, e `C0`, `C1`, `C2`, ..., são *SingleCommand* da linguagem Triangle. O caso `else:` é obrigatório e é forçosamente o último a aparecer.

A semântica é a seguinte. A expressão `E` é avaliada. Seja x o valor da expressão. Subsequentemente será executado um e um só *SingleCommand* correspondente ao *IntegerLiteral* cujo valor é x. Se não existir, é executado o comando `C0`, correspondendo ao caso `else`. (o equivalente ao 'default' num switch em C ou Java).

Como exemplo, o seguinte programa em Triangle espera que o utilizador introduza um número inteiro e escreve no ecrã `28` (se o inteiro introduzido for 2), `30` (se o inteiro introduzido for 4, 6, 9, ou 11), ou `31` (se o inteiro introduzido for outro valor).

```
let
   var month: Integer;
   var days: Integer;
   var leap: Boolean
in
   begin
      leap := false;
      getint(var month);
      case month of
         2: days := if leap then 29 else 28;
         4: days := 30;
         6: days := 30;
         9: days := 30;
        11: days := 30;
      else: days := 31;
      putint(days)
   end
```

5 Prazo e entrega do trabalho

Pontuação máxima: 40 pontos.

Prazo: 2ª feira, 18/Mai/2020 até às 23:59h.

Entrega: O trabalho deve ser entregue através da tutoria num único ficheiro ZIP que contenha a pasta `Triangle` e todas as suas sub-pastas. No caso de grupos com mais do que 1 aluno, apenas 1 dos elementos do grupo deve submeter o trabalho.