Processamento de Linguagens
Licenciatura em Engenharia de Sistemas Informáticos
2023/2024

Trabalho Prático 02

1 Introdução

Com este projeto pretende-se que os alunos da UC de Processamento de Linguagens adquiram experiência na definição de analisadores léxicos e sintáticos, bem como na definição de ações semânticas que traduzem as linguagens implementadas.

O processo para a realização deste trabalho prático deverá passar pelas seguintes fases:

1. especificar a gramática concreta da linguagem de entrada;
2. construir um reconhecedor léxico (recorrendo à biblioteca *l*ex) para reconhecer os símbolos terminais identificados na gramática, e testar esse reconhecedor com alguns exemplos de palavras da linguagem de entrada.
3. construir um reconhecedor sintático (recorrendo à biblioteca *y*acc) para reconhecer a gramática concreta, e testar esse reconhecedor com alguns exemplos frases da linguagem de entrada.
4. planear uma árvore de sintaxe abstrata para representar a linguagem de entrada, e associar ações semânticas de tradução às produções da gramática de forma a construir a correspondente árvore de sintaxe abstrata.
5. desenvolver o gerador de código que produza a resposta solicitada, através da avaliação da árvore de sintaxe abstrata.

Nesse sentido, o que se propõe é a implementação de uma aplicação para o processamento de uma linguagem funcional, designada por *Linguagem Funcional do Cávado e do Ave*, abreviadamente `FCA`. A aplicação implementada deverá ser capaz de ler um ficheiro de texto com um programa escrito usando a linguagem `FCA`, descrita na secção 3.1, e deverá executar os comandos nele contidos. Opcionalmente, a aplicação gera um ficheiro de texto com o código C correspondente.

Na secção seguinte são apresentadas as regras deste trabalho prático. Na secção 3 é apresentado o enunciado do problema proposto para este trabalho prático.

2 Regras

- O trabalho tem carácter obrigatório para aprovação à unidade curricular, deve ser realizado em grupo de até 3 elementos;
- O relatório deve introduzir o problema a ser resolvido, apresentar a abordagem seguida na sua resolução, quais os objectivos atingidos e quais os problemas encontrados. No relatório devem ser salientados todos os pontos que os alunos achem que poderão valorizar o seu trabalho em relação aos requisitos como, por exemplo, funcionalidades adicionais. Também deverá conter uma secção que descreva de que forma é que a aplicação foi testada.
- O trabalho contempla uma apresentação e defesa individual em horário a agendar pelo docente. Esta defesa tem aprovação obrigatória, sendo que a falta à defesa corresponderá à não entrega do trabalho pelo aluno (i.e. avaliação de zero valores); Será agendado um horário com cada grupo para a apresentação do trabalho.

  Devem ser colocadas todas as referências consultadas durante a elaboração do trabalho (bibliográficas ou mesmo consultas *online*);
- Durante as aulas dedicadas aos trabalhos poderá ser solicitado aos alunos que apresentem o trabalho desenvolvido até esse momento.
- A data de entrega final é aquela que foi estabelecida no início do semestre.
- Não serão aceites entregas ou melhorias após a data definida neste enunciado. Não serão aceites entregas ou melhorias nas épocas de exame (este trabalho apenas é válido para a avaliação da época em que é lançado).
- O esclarecimento de dúvidas acerca deste documento pode originar a publicação de novas versões.

3 Enunciado

Neste trabalho prático pretende-se que os alunos implementem uma aplicação em `Python`, usando a bilioteca `PLY`, que interprete uma linguagem capaz de especificar algumas instruções que habitualmente encontramos em linguagens de programação funcionais.

A aplicação a desenvolver deverá começar por ler um ficheiro de texto (com extensão `.fca`, para funcional do Cávado e do Ave) contendo uma sequência de comandos de especificação da linguagem, aplique esses comandos de forma a calcular o resultado pretendido. O resultado de processar o ficheiro de texto `entrada.fca` é apresentado ao utilizador no terminal (opcionalmente, o programa poderá gerar um ficheiro com a respetiva implementação em código C). Caso não seja apresentado o ficheiro de entrada, os comandos devem ser lidos do terminal à medida que o utlizador os vai inserindo.

3.1 Linguagem FCA

A linguagem `FCA` é definida por uma sequência de instruções, seguidas pelo separador ponto e vírgula. Uma instrução poderá ser: uma instrução de escrita, uma instrução de saída, ou uma atribuição.

Uma instrução de escrita, comando `ESCREVER`, envia para a consola o valor como parâmetro.

Uma atribuição passa por associar a uma variável uma expressão aritmética (conforme estudamos nas aulas), ou o resultado de leitura de um valor introduzido pelo utilizador `ENTRADA()`.

Nos exemplos temos que os comandos são escritos em maiúsculas, no entanto podem ser utilizadas maiúsculas e minúsculas. Por outro lado, assume-se que para cada comando temos uma versão compacta com as três primeiras letras.

A. atribuição a uma variável

Uma atribuição passa por atribuir a uma variável: uma expressão aritmética, o resultado de leitura de um valor introduzido pelo utilizador, ou a geração aletória de um valor.

Os identificadores de variáveis devem começar obrigatoriamente por uma letra minúscula (ou *underscore*). Nas posições seguintes poderão ter qualquer combinação de letras, dígitos e *underscore*. Pode terminar com um ponto de interrogação ou um sinal de exclamação.

```
tmp_01 = 2*3+4 ;
a1_ = 12345 - (5191 * 15) ;
idade_valida?  = 1;
mult_3!  = a1_ * 3 ;
```

Uma expressão aritmética poderá ser atribuída a uma variável. Devem completar as expressões regulares usadas nas aulas outros operadores como subtração, divisão (inteira), possibilidade de ter simétrico de um número.

Na sequência de instruções apenas o valor do cálculo da última instrução é que será devolvido (no caso do programa principal mostra o valor). Na sequência de instruções anterior, é devolvido o valor de calcular $a1\_ * 3$.

B. instruções de I/O e strings

Para facilitar a elaboração de mensagens de saída temos a representação de strings (texto) entre aspas. Temos o operador de concatenação de strings `<>`. Segue um exemplo da utilização de instruções de escrita:

```
ESCREVER(valor); -- conteúdo de valor é apresentado
ESCREVER(365 * 2); -- 730
ESCREVER("Ola Mundo"); -- Olá, Mundo!
curso = "ESI";
```

```
ESCREVER("Olá, "<> curso); -- Olá, ESI
```

Os comentários apenas de uma linha são precedidos por −− (todo o texto seguinte é ignorado). No caso dos comentários em mais do que uma linha, temos os seguintes operadores {− ... −}. Podemos ainda recorrer à interpolação de strings para compor a uma string a partir do conteúdo de outras variáveis.

```
{- exemplo interpolação de strings
Olá, EST IPCA!-}
escola ="EST";
inst = "IPCA";
ESCREVER ("Olá, #{escola}  #{inst}!");
```

O comando de escrita apenas aceita um parâmetro. Existem ainda funções para a entrada de valores:

```
valor = ENTRADA();
ate10 = ALEATORIO(10);
```

O valor a atribuir a uma variável poderá ser lido da consola, ou então um número aleatório entre zero e o número inteiro indicado (no caso apresentado, 10).

C. declaração de funções

A função tem dois modos de declaração: apenas numa linha, quando a declaração termina com ,: (admitindo apenas uma instrução). e a versão em mais do que uma linha, onde são admitidas várias instruções, até seja indicada a palavra reservada `FIM`. O valor devolvido pela função é o resultado de calcular a última instrução dessa sequência. É possível declarar uma função conforme o exemplo:

```
FUNCAO soma(a,b),:  a+b ;
FUNCAO soma2(c) :
  c = c+1 ;
  c+1 ;
FIM
seis = soma(4,2);
oito = soma2(seis);
```

Esta sequência de instruções devolve o cálculo da última instrução, ou seja, 8. De salientar que dentro de uma função não é possível usar instruções de I/O (leitura e escrita).
Consideremos agora o seguinte exemplo para o cálculo de áreas de retangulos e quadrados:

```
FUNCAO area_retangulo(a, b):
  a * b;
FIM
```

```
FUNCAO area_quadrado(a):
  area_retangulo(a, a);
FIM
a = area_retangulo(10, 20);
b = area_quadrado(30);
```

Em alternativa, estas duas funções podem ser distinguidas através do número de parâmetros (neste caso a função está definida numa única linha).

```
FUNCAO area(a,b),:  a*b ;
FUNCAO area(c),:  area(c, c);
d = area(10, 20);
e = area(30);
```

Podemos ainda definir uma função por ramos, definindo o resultado para cada situação

```
FUNCAO fib( 0 ),:  0 ;
FUNCAO fib( 1 ),:  0 ;
FUNCAO fib( n ):
  a = fib(n-1);
  b = fib(n-2);
  a + b;
FIM
fib5 = fib(5);
```

D. listas

A linguagem admite a especificação de listas, em que os elementos podem ser todos inteiros.

```
lista = [ 1, 2, 3 ] ;
ESCREVER( lista ); -- [1,2,3]
vazia = [] ;
```

Consedere que estão definidas as funções seguintes funções sobre listas: `map` que dada uma função f aplica f a cada elemento da lista; `fold` que é usada para calcular o resultado de acumular os resultados de aplicar uma função g aos elementps da lista

```
FUNCAO mais2( x ),:  x + 2 ;
FUNCAO soma( a, b ),:  a + b ;

lista1 = map( mais2, [] ); -- []
lista2 = map( mais2, [ 1, 2, 3 ] );
  -- [ mais2(1),mais2(2),mais2(3)] = [3,4,5]
lista3 = fold( soma, [ 1, 2, 3 ], 0 );
```

```
-- = soma( 1, soma(2, soma ( 3, 0)))
-- = soma( 1, soma(2, 3 ))
-- = soma( 1, 5)
-- = 6
```

É possível definir uma função tendo em conta os seguintes casos de valores de uma lista: [ ] para o caso de lista vazia; $x : xs$ para o caso em que é uma lista, não vazia, sendo o primeiro elemento x e a lista com os restantes estão em xs. Exemplo do cálculo da soma dos elementos de uma lista:

```
FUNCAO somatorio( [] ),:  0 ;
FUNCAO somatorio( x:xs ),:  x + somatorio(xs) ;
resultado = somatorio([1,2,3]);
```

E. outras instruções

Cada grupo poderá ainda sugerir uma implementação e integração de outras funcionalidades, tais como: uso do condicional (SE); operadores lógicos: `/\,\/`, `NEG` .

3.2 resultado

Desenvolver o gerador de código que produza a resposta solicitada, através da avaliação da árvore de sintaxe abstrata para:

- executar as instruções no ficheiro de entrada, permitindo ver o correspondente resultado;
- criar um ficheiro em C, com um programa equivalente à sequência de instruções.

A avaliação deste trabalho terá em conta:

- a qualidade do reconhecedor léxico e da gramática implementados;
- a estrutura da Árvore Abstrata de Sintaxe;
- a diversidade de operadores da linguagem apresentada no enunciado;
- a organização e qualidade do código Python desenvolvido;
- a possibilidade de geração de código C.

3.3 Dúvidas

Quando o enunciado não for explícito em relação a um requisito específico, os alunos podem optar pela solução que parecer mais adequada, fazendo a sua apresentação e justificação no relatório. Dúvidas adicionais devem ser colocadas ao docente pessoalmente ou por *email*.