

# Processamento de Linguagens e Compiladores

LMCC, Universidade do Minho

Ano lectivo 2005/2006

Trabalho Prático N°2

## Um Interpretador da Linguagem BC--

Grupo: 55436 LESI Ana  
40000 MCC Tó  
33333 MCC Zé

Neste trabalho prático pretende-se que os alunos implementem um interpretador para uma linguagem muito simples: A linguagem com nome BC--. O programa a desenvolver deverá utilizar as técnicas leccionadas nesta cadeira, bem como da disciplina de Cálculo de Programas: *gramáticas independentes de contexto, parsing, geração de programas, árvores de sintaxe abstracta e funções que efectuam recursividade (travessias) em árvores* (e.g., catamorfismos).

---

**Entrega da Trabalho:** dia 6 de Junho

**Constituição dos Grupos:** Os grupos terão obrigatoriamente **3 elementos**

**Linguagem de Prog.:** Lex/Yacc e C

---

O enúnciado do trabalho prático está escrito em **literate Programming** usando o sistema **NoWeb**<sup>1</sup>. Isto é, pode ser interpretado como um documento  $\text{\LaTeX}$  ou como um programa numa qualquer linguagem de programação. Resolva este trabalho prático neste mesmo ficheiro de modo a implementar o programa e a sua documentação.

## 1 A linguagem BC--

A linguagem BC-- é uma linguagem para definir expressões aritméticas muito simples usuais em qualquer linguagens de programação. Esta linguagem é ainda utilizada por um comando

---

<sup>1</sup>O Sistema NoWeb está disponível no seguinte endereço: <http://www.eecs.harvard.edu/nr/noweb/>.

do sistema operativo Unix/Linux que implementa uma máquina de calcular<sup>2</sup>. Quando esta máquina de calcular foi introduzida em Unix, o *bc* significava *Basic Calculator*. Porém, esta linguagem/ferramenta foi evoluindo e actualmente a linguagem *bc* é já um razoável sub-conjunto da linguagem C. Neste trabalho vamos considerar a definição original da linguagem, e daí a designarmos por BC-- . Uma frase concreta nesta linguagem tem a seguinte forma:

```
aux = 3 + 4 ;
print aux ;
b = aux * 4 ;
print "\n b vale: " , b ;
quit ;
```

Como se verifica neste exemplo, uma frase em BC-- é uma sequências de instruções, seguidas do separador ;. Uma instrução é uma de três coisas: uma atribuição, uma instrução de *print* ou ainda uma instrução de saída. Uma atribuição é definida atribuindo a uma variável uma expressão aritmética. Para não complicar o trabalho consideram-se apenas os operadores +, -, \* e /. Uma instrução de *print* envia para a saída os valores que estão definidos na lista que é seu argumento. este valores pdoem ser expressões aritméticas, ou *string*. A instrução de saída é denotada pela palavra reservada *quit* e deverá ocorrer sempre como última instrução da frase.

Uma frase BC-- mais elaborada é:

*<exemplo.bc>*≡

```
a = 3 + 4 * 5 ;
print a , "\n" ;
b = a * 4      ;
c = b --      ;
c += a        ;
print "valor de b=" , b + 4 , "\nvalor de c=",c ;
quit ;
```

---

<sup>2</sup>Para saber mais detalhes desta linguagem/ferramenta, utilize os manuais do sistema operativo, isto é, veja o resultado de `man bc`

Esta frase é também um programa *bc* válido. O resultado de processar esta frase em *bc* é:

```
[jas@hell plc]$ bc exemplo.bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
23
valor de b=95
valor de c=115
```

Neste trabalho prático, pretende-se construir um programa que dada uma frase nesta linguagem interprete as várias expressões aritméticas e produza os resultados

O Interpretador deve ainda detectar e sinalisar erros léxicos, sintáctico e semânticos. As regras semânticas de *BC--* são:

- Num programa em *BC--*, não podem ser usados identificadores que não tenham sido declarados anteriormente.
- Não são permitidos definições duplicadas do mesmo identificador.

## 2 Desenvolvimento do Projecto

Na realização deste projecto, utilizamos o sistema LexYacc [Nie91]. Estes sistemas ...

Utilizamos ainda a linguagem C [KR88], uma vez que estes sistemas se baseiam nesta linguagem. Para efectuar a gestão de memória do processador desenvolvido em C, utilizamos o gestor de memória do Boehm-Demers-Weiser. Este gestor de memória para programas C e C++ está disponível em: [http://www.hp1.hp.com/personal/Hans\\_Boehm/gc/](http://www.hp1.hp.com/personal/Hans_Boehm/gc/)

É bastante fácil utilizar, bastando para tal ...

## 3 Gramáticas Independentes do Contexto

A gramática independente de contexto que define a *estrutura concreta* de uma frase na linguagem SEJA é a seguinte:

$$\begin{aligned} G_{concreta} &= (T, N, S, P), \text{ com} \\ T &= \{int, id, '+', \dots\}, \\ N &= \{BC, \dots\}, \\ S &= BC, \\ P &= \left\{ \begin{array}{l} prodBC : BC \rightarrow \dots \\ , p_1 : \dots \rightarrow \dots \\ , \dots \\ \end{array} \right\} \end{aligned}$$

A estrutura abstracta da linguagem é definida pela seguinte gramática abstracta:

$$\begin{aligned}
 G_{abstracta} &= (T, N, S, P), \text{ com} \\
 T &= \{int, id\}, \\
 N &= \{BC, \dots\}, \\
 S &= BC, \\
 P &= \{ \text{ProdBC} : BC \rightarrow \dots \\
 &\quad , \dots \quad Expr \rightarrow \dots \\
 &\quad , \dots \\
 &\quad \}
 \end{aligned}$$

### 3.1 Árvore Abstracta de BC-- em C

Para construirmos a árvore abstracta que representa abstractamente a linguagem BC--, utilizamos a ferramenta `Gram2C`. Esta ferramenta tem as seguintes características:

- ...
- ...

A especificação da árvore abstracta de BC-- em `Gram2C` obtém-se facilmente a partir da gramática  $G_{abstracta}$  apresentada na secção anterior.

$\langle ArvoreAbst.stp \rangle \equiv$

### 3.2 Parsing da Linguagem BC-- em YACC

O parser para a linguagem BC-- obtém-se escrevendo a gramática concreta na notação do YACC [Nie91]. As funções semânticas usadas nos acções semânticas do YACC são os construtores da árvore abstracta. Assim temos o seguinte parser:

$\langle Parser.y \rangle \equiv$   
 $\langle Gramática Concreta da Linguagem BC- em YACC \rangle$

```
<Gramática Concreta da Linguagem BC- em YACC>≡  
%%  
seja : ...
```

## 4 Interpretador da Linguagem BC-

Um dos objectivos do nosso programa é efectuar o cálculo de expressões aritméticas, i.e., interpretar essas expressões. Como o resultado do parser de uma frase é uma árvore do tipo BC, temos agora de construir uma função que recursivamente faz uma travessia nessa árvore e sintetiza o valor da expressão pretendida<sup>3</sup>. De seguida apresentamos esta função em C:

```
<Processador.c>≡  
<Processador da Linguagem Seja em C>
```

```
<Processador da Linguagem Seja em C>≡  
#include <stdio.h>  
  
...
```

## 5 A makefile

A makefile para produzir o programa desejado apresenta-se a seguir

---

<sup>3</sup>Isto é, esta função é um catamorfismo sobre o tipo de dados BC

```
<makefile>≡
LEX = /usr/bin/lex
YACC = /usr/bin/yacc -d
GRAM2C = ???/stp
CC = gcc -O2

bc-- : ...
      $(CC) -o bc-- ...

...
```

## 6 Extras ao trabalho prático

1. **Pretty Printing** Escreva uma função `show` que dada a árvore abstracta (do tipo `BC`) a mostra na sua notação concreta. Isto é, realiza a tarefa contrária a um parser, pois da representação abstracta contrói uma concreta. Esta tarefa designa-se usualmente por *unparsing* ou *pretty printing*.

Utilize ainda esta função para mostrar ao utilizador onde ocorrem os erros semânticos. Por exemplo, dado este programa ao processador

```
a = 3 * 4;
a = 2 ;
print c ;
quit ;
```

Será mostrado ao utilizador o seguinte programa:

```
a = 3 * 4;
a = <- Erro Semântico: Variavel já definida -> 2
print c <- Erro Semântico: Variavel não definida ->
quit ;
```

Caso nenhuma erros semântico ocorra, então é produzido exactamente o mesmo programa dado como entrada.

2. **Geração de Código** Implementação de um compilador: O programa proposto implementa um interpretador da linguagem. Propõe-se como extra desenvolver um compilador para o assembly MSP e utilizar o ambiente winMSP para interpretar o assembly gerado pelo compilador.

Para tal, defina a gramática asbtracta do código MSP a ser produzido, converta-a num tipo de dados `C` e escreva a função `show` que transforma a estrutura abstracta na representação concreta do MSP.

## Referências

- [KR88] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language, Second Edition*. Prentice Hall, Inc., 1988.
- [Nie91] Thomas Niemman. *A Compact Guide to Lex & Yacc*. epaperpress.com, 1991.