

[MAC0211] Laboratório de Programação I
Aula 16
Revisão para a Prova 1
———
Análise Léxica

Alair Pereira do Lago

DCC-IME-USP

23 de abril de 2015

Revisão para a prova 1 (dia 07/05)

Matéria da prova

- ▶ Introdução a história e arquitetura dos computadores
- ▶ Linguagem de montagem
- ▶ Bibliotecas estáticas × bibliotecas dinâmicas
- ▶ Interpretadores de comandos
- ▶ Arquivos, *pipes* e filtros
- ▶ Projeto de interfaces
- ▶ Automação de compilação

Introdução a história e arquitetura dos computadores (aulas 1–2)

Conhecimentos esperados

- ▶ Principais características das 4 gerações de computadores apresentadas em aula
- ▶ Arquitetura de Von Neumann
- ▶ Arquitetura da família de processadores Intel x86
- ▶ O ciclo de busca e execução

Linguagem de montagem (aulas 2–8)

Conhecimentos esperados

- ▶ Vantagens do uso
- ▶ Estrutura geral das instruções
- ▶ Instruções para a movimentação de dados (ex.: mov, xcgh)
- ▶ Instruções aritméticas (ex.: add, dec, mul, idiv)
- ▶ Instruções lógicas (ex.: and, or, xor, not)
- ▶ Instruções para saltos (ex.: jmp, jnz, je, cmp)
- ▶ Instruções para o uso da pilha (ex.: push, pop, call, ret)
- ▶ Implementação de funções e chamadas a funções (incluindo chamadas a funções de C)
- ▶ Chamadas ao sistema operacional
- ▶ Montador NASM: estrutura do código-fonte, declaração de variáveis e constantes

Bibliotecas estáticas × bibliotecas dinâmicas (aula 9)

Conhecimentos esperados

- ▶ Características gerais de uma biblioteca (assunto relacionado: Interfaces, dado nas aulas x e y)
- ▶ Vantagens e desvantagens das bibliotecas estáticas
- ▶ Vantagens e desvantagens das bibliotecas dinâmicas

Interpretadores de comandos (aulas 10–12)

Conhecimentos esperados

- ▶ Conceitos relacionados a *shells*
- ▶ Programas utilitários mais comuns (ex.: `ls`, `cd`, `rm`, `find`, `less`, `grep`)
- ▶ *Bash*
 - ▶ operações de redirecionamento e *pipelines*
 - ▶ *scripts*
 - ▶ variáveis de ambiente
 - ▶ construtores de programação (ex.: comandos `while`, `for`, `if`, `select`)
 - ▶ argumentos passados via linha de comando
 - ▶ funções

Arquivos, *pipes* e filtros (aulas 12–13)

Conhecimentos esperados

- ▶ Classificações de arquivos
- ▶ Arquivos de dispositivos
- ▶ *Streams*
- ▶ Comunicação entre programas via *pipes*
- ▶ Programas implementados como filtros

Projeto de interfaces (aulas 13–14)

Conhecimentos esperados

- ▶ Questões a serem consideradas no projeto de software
 - ▶ Interfaces
 - ▶ Ocultação de informações
 - ▶ Gerenciamento dos recursos
 - ▶ Tratamento de erros
- ▶ Princípios para o desenvolvimento de boas interfaces
 - ▶ Ser unificada, simples, suficiente, genérica, estável
 - ▶ Ocultação dos detalhes de implementação
 - ▶ Conjunto ortogonal pequeno de funções
 - ▶ Não sair do alcance do usuário
 - ▶ Fazer uma mesma coisa igual em todos os lugares

Automação de compilação (aulas 14–15)

Conhecimentos esperados

- ▶ Conceito de construção de software
- ▶ GNU Make
 - ▶ Arquivos *Makefiles* e seu funcionamento
 - ▶ Regras
 - ▶ Variáveis
 - ▶ Alvos falsos
 - ▶ Regras implícitas

Alguns exercícios

Como um exercício para as matérias mais “práticas” da P1, considere a seguinte prova de LabProg I aplicada em 2001:

- ▶ <http://www.ime.usp.br/~kon/MAC211/provasExemplo/P1/>

Mudando de assunto...

[Aula passada] Processamento de Linguagens de Programação

- ▶ **Sintaxe de uma linguagem**: especifica **como** os programas de uma linguagem são construídos
- ▶ **Semântica de uma linguagem**: especifica **o quê** os programas significam

Exemplo

Suponha que datas são construídas a partir de dígitos representados por *D* e o símbolo /, da seguinte maneira: **DD/DD/DDDD**

- ▶ De acordo com essa sintaxe, **01/02/2013** é uma data
- ▶ O dia a que essa data se refere não é identificado pela sintaxe. Por exemplo, no Brasil, essa data corresponde ao dia primeiro de fevereiro de 2013. Entretanto, nos EUA, ela corresponde ao dia dois de fevereiro de 2013
- ▶ Portanto, **podemos ter mais de uma semântica associada à uma mesma sintaxe**

[Aula passada] Processamento de Linguagens de Programação

Etapas envolvidas em um processo de compilação:

- ▶ **Análise Léxica** – responsável por identificar os itens léxicos (palavras, números, símbolos, etc.) no código-fonte do programa
- ▶ **Análise Sintática** – responsável por identificar a estrutura sintática do programa e construir uma estrutura de dados (normalmente uma árvore sintática) para representar este programa em memória
- ▶ **Análise Semântica** – responsável por definir o que significa cada comando e gerar o código correspondente

[Aula passada] Processamento de Linguagens de Programação

Neste curso abordaremos formalmente apenas a **Análise Léxica**

Processamento de programas é visto em:

- ▶ **MAC 316 – Conceitos de Linguagens de Programação** (construção de um interpretador)
- ▶ **MAC 414 – Linguagens Formais e Autômatos** (análise sintática em profundidade)
- ▶ **MAC 410 – Introdução à Compilação** (construção de um compilador)

[Aula passada] Expressões e notações

- ▶ Expressões como $a + b * c$ são usadas há séculos e foram o ponto de início para o projeto de linguagens de programação
- ▶ Linguagens de programação usam um misto de notações. Por exemplo:
 - ▶ $a + b$ – operador aparece entre os operandos
 - ▶ $\text{sqrt}(a)$ – operador aparece antes do operando
 - ▶ $a++$ – operador aparece depois do operando
- ▶ **Notação infixa**: um operador binário é escrito entre os seus dois operandos (ex.: $a + b$)
- ▶ **Notação prefixa**: um operador binário é escrito antes dos seus dois operandos (ex.: $+ab$)
- ▶ **Notação pós-fixa**: um operador binário é escrito depois dos seus dois operandos (ex.: $ab+$)

[Aula passada] Notação infixa

Vantagem dessa notação:

- ▶ É a mais “familiar” das notações e, por isso, mais fácil de se ler

Desvantagem dessa notação:

- ▶ Ambiguidade na decodificação. Exemplo: a expressão pode $a + b * c$ conduzir a duas decodificações diferentes:
 - ▶ a soma de a e $b * c$
 - ▶ a multiplicação de $a + b$ e c
- ▶ Desambiguação é feita com:
 - ▶ precedência de operadores
 - ▶ associatividade de operadores de mesma precedência
 - ▶ parênteses

[Aula passada] Notação infixa

Exemplos:

- ▶ Operadores associativos-à-esquerda: $+$, $-$, $*$ e $/$

Correto: $4 - 2 - 1 = (4 - 2) - 1 = 2 - 1 = 1$

Errado: $4 - 2 - 1 = 4 - (2 - 1) = 4 - 1 = 3$

- ▶ Operador associativo-à-direita: exponenciação

$$2^{3^4} = 2^{(3^4)} = 2^{81}$$

[Aula passada] Notação prefixa

Uma expressão em notação prefixa é escrita do seguinte modo:

- ▶ A notação prefixa de uma constante ou variável é a própria constante ou variável
- ▶ A aplicação de um operador **op** às sub-expressões E_1 e E_2 é escrita na notação prefix como **op** E_1 E_2

Vantagens dessa notação:

- ▶ É fácil de decodificar fazendo uma varredura da esquerda para a direita na expressão
- ▶ Dispensa o uso de parênteses (não há ambiguidade na identificação dos operandos de um operador)

Exemplos

- ▶ $* + 20\ 30\ 60 = * 50\ 60 = 3000$
- ▶ $* 20 + 30\ 60 = * 20\ 90 = 1800$

[Aula passada] Notação pós-fixa (= *notação polonesa*)

Uma expressão em notação pós-fixa é escrita do seguinte modo

- ▶ A notação pós-fixa de uma constante ou variável é a própria constante ou variável
- ▶ A aplicação de um operador **op** às sub-expressões E_1 e E_2 é escrita na notação pós-fixa como $E_1 E_2 \text{ op}$

[Aula passada] Vantagens dessa notação:

- ▶ São fáceis de se avaliar de forma automatizada com o auxílio de uma estrutura de pilha
- ▶ Dispensa o uso de parênteses (não há ambiguidade na identificação dos operandos de um operador)

Exemplos

- ▶ $20\ 30\ +\ 60\ * = 50\ 60\ * = 3000$
- ▶ $20\ 30\ 60\ +\ * = 20\ 90\ * = 1800$

Notação “*mixfixa*”

Operações especificadas por uma combinação de símbolos não se enquadram na classificação infixa, prefixa e pós-fixa.

Exemplo

if $a > b$ then a else b

Análise Léxica (na prática)

Exemplo: Calculadora HP

- ▶ Calcula o valor de expressões aritméticas na **notação pós-fixa** (também chamada de **notação polonesa**)
- ▶ Operadores aceitos: +, -, *, /
- ▶ Outros comandos:
 - ▶ '=' – mostra o resultado (topo da pilha)
 - ▶ 'l' – lista os valores na pilha
 - ▶ 'c' – limpa a pilha
 - ▶ 'f' – sai do programa
- ▶ Ver código da implementação disponível no Paca

Análise Léxica (na prática)

Exemplo: Calculadora recursiva (notação infixa)

- ▶ Exemplo de utilização:

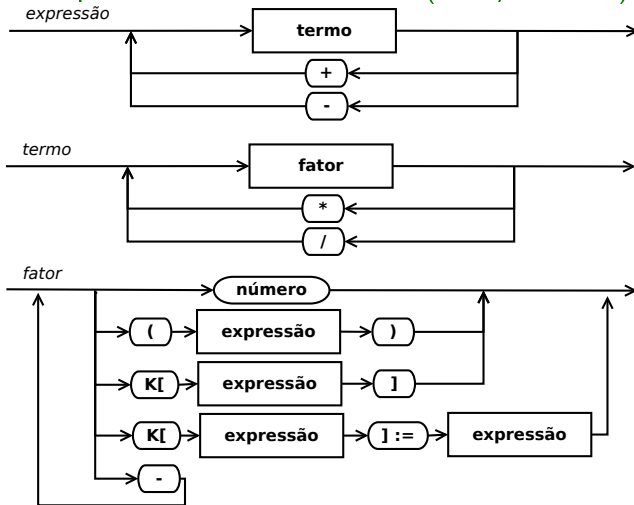
```
(2*(3+(5*2)));  
k[0] := (2+(4/20))-18.32;  
k[1] := 3.1415926538*2;  
(123 + k[1])*2 + k[0];  
2+(K[3]:=9);
```

- ▶ Gramática (informalmente definida):

```
<programa> ::= <programa> <expr> ; | <expr> ;  
<expr>    ::= <expr> + <termo> | <expr> - <termo> | <termo>  
<termo>   ::= <termo> * <fator> | <termo> / <fator> | <fator>  
<fator>   ::= NUMERO | -<fator> | (<expr>) |  
                K[<expr>] | K[<expr>] := <expr>
```

Análise Léxica (na prática)

Exemplo: Calculadora recursiva (notação infixa)



Análise Léxica (na prática)

Exemplo: Calculadora recursiva (notação infixa)

- ▶ Operadores aceitos: +, -, *, /
- ▶ Outros comandos:
 - ▶ 'k[i]' ou 'K[i]' – variável
 - ▶ ':= ' – atribuição de valor pra uma variável
 - ▶ ';' – separador de expressões
 - ▶ 'q', 'Q', 'f' ou 'F' – sai do programa
- ▶ Ver código da implementação disponível no Paca

Bibliografia e materiais recomendados

- ▶ Capítulo 2 do livro *Programming Languages – Concepts & Constructs*, de Ravi Sethi
- ▶ Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
<http://www.ime.usp.br/~kon/MAC211>

Cenas dos próximos capítulos...

Nas próximas aulas:

- ▶ Processamento de macros
- ▶ Expressões regulares