## Instituto de Computação da UNICAMP

## Disciplina MC202: Primeiro Semestre de 2013

## Laboratório Nº 10

*Prof. Tomasz Kowaltowski*

---

O objetivo desta tarefa de laboratório é a implementação e teste de *árvores digitais* (*ADs*), também conhecidas como *tries*. A implementação deverá seguir extatamente o exposto em aula, com a opção mais simples para os nós, conforme indicado na transparência 259, relativa ao alfabeto padrão de 26 letras minúsculas.

Será seguida novamente a idéia de um tipo abstrato de dados realizado através de um par de arquivos `trie.h` e `trie.c`. O arquivo `trie.h` contém a especificação exata das funções de interface.

A implementação será testada por um programa `principal.c` que usará uma AD para administrar um conjunto de cadeias. O programa principal aceita os seguintes comandos:

| | |
|---|---|
| **c** | cria e devolve uma AD vazia (v. obs. 4) |
| **i** ***s*** | insere a cadeia *s* na AD (*s* pode ser vazia) |
| **v** ***s*** | verifica se a cadeia *s* está na AD (*s* pode ser vazia) |
| **r** ***s*** | remove a cadeia *s* da AD (*s* pode ser vazia) |
| **p** | visita as cadeias da AD em ordem alfabética |
| **n** | imprime o número de nós da AD |
| **a** | imprime a altura da AD |
| **k** | imprime imprime o número de cadeias na AD |
| **l** | libera a memória dinâmica ocupada pela AD |
| **h** | imprime o resumo dos comandos |
| **x** ou **q** | encerra a interpretação |
| **#** | linha de comentário |

**Observações:**

1. A representação da AD está escondida do usuário que deve enxergá-la apenas como um *tipo abstrato de dados (TAD)*. Isto é feito no arquivo `trie.h` definindo o tipo `Trie` como um apontador para o tipo `void`. A declaração do tipo usado para implementação ficará escondida no arquivo `trie.c` onde as conversões de tipo apropriadas (*casting*) deverão ser efetuadas, se necessário (algumas são automáticas).

2. É fornecida uma versão incompleta do arquivo `trie.c`. Ele contém algumas funções prontas e sugestões de algumas funções auxiliares. É o único arquivo a ser submetido.

3. A cadeia a ser inserida ou removida poderá ser vazia (isto é, de comprimento zero); neste caso, a marca de *fim* de cadeia da raiz da AD indicará este fato.

4. A fim de facilitar a implementação, a árvore digital vazia (AD inicial) deverá ser criada com um nó raiz com todas as subárvores vazias e a marca de *fim* de cadeia falsa.

5. Os testes para a tarefa estão divididos em três grupos:

    - Grupo 1 (11, 12, 13, 14, 15): são testadas todas as funções, exceto *removeAD* e *percorreAD*;
    - Grupo 2 (21, 22, 23, 24, 25): são testadas todas as funções, exceto *percorreAD*;
    - Grupo 3 (31, 32, 33, 34, 35): são testadas todas as funções.

6. A tarefa 10a executará todos os testes e sua nota máxima será 12,0; a tarefa 10b executará os testes dos grupos 1 e 2 e sua nota máxima será 10,0; a tarefa 10c executará apenas os testes do grupo 1 e sua nota máxima será 7,0.

7. Devem ser utilizadas as macros *MALLOC* e *FREE* fornecidas com o pacote *balloc*. A sua implementação não pode utilizar as operações habituais de alocação.

8. Algumas funções podem ser facilmente implementadas sem usar recursão: *consultaAD*, *insereAD*, *removeAD*. A decisão fica livre. (No caso de *removeAD* seria necessária uma pilha.) As outras funções têm implementação mais simples com o uso da recursão.

9. Podem ser declaradas outras funções auxiliares.

10. O número máximo de submissões de cada tarefa é 10.

---