MC202 — ESTRUTURAS DE DADOS

Lab 08 — Árvore Binária de Busca (ABB)

Problema

Godofredo é dono de uma livraria e decidiu que era hora de modernizar suas operações. Tendo em vista que um dos principais motivos de frustração para seus clientes é o tempo gasto verificando se a loja possui um livro desejado, Godofredo concluiu que seu primeiro passo seria disponibilizar um sistema de busca digital para agilizar o processo e permitir que os clientes obtenham essa informação via Internet.

Para tornar o processo de busca mais rápido e agradável ao usuário, ele decidiu que a melhor opção seria utilizar um sistema de autocompletar em que, à medida que o usuário digita o nome do livro buscado, o sistema lista em ordem lexicográfica os livros disponíveis que possuem o prefixo já fornecido.

Neste laboratório, sua tarefa é implementar uma estrutura de dados que atenda às necessidades de Godofredo, ou seja, que permita a realização das seguintes operações de forma eficiente:

- Adicionar e remover livros no sistema.
- Listar todos os livros cujos nomes sejam compatíveis com um dado prefixo, em **ordem lexicográfica**.

Entrada

A entrada é composta por uma sequência de linhas. Cada linha representa uma operação no formato `[op][string]`, sendo `[op]` um único caractere indicando qual é a operação e `[string]` uma string de tamanho n, $1 \leq n < 80$, contendo apenas letras minúsculas e espaços, que serve de parâmetro para a operação. As operações são:

- Adicionar livro:
  `[op]` = `'+'` e `[string]` é o nome do livro a ser adicionado;
- Remover livro:
  `[op]` = `'-'` e `[string]` é o nome do livro a ser removido;
- Listar sugestões para autocompletar:
  `[op]` = `'$'` e `[string]` indica o prefixo para o qual se deve listar as sugestões;
- Terminar programa:
  `[op]` = `'#'` e `[string]` = `"fim"`, aparecendo sempre na última linha.

Você pode supor que não serão adicionados livros repetidos e que a remoção sempre indica o nome exato de um livro existente. Também, suponha que o prefixo sempre tem pelo menos um caractere. O conjunto de sugestões para o mecanismo de autocompletar pode ser vazio.

Saída

Para cada chamada da operação '`$`' seu programa deve imprimir uma linha com a frase `Sugestoes para prefixo "[string]":` e mais uma linha com o nome de cada livro compatível com o prefixo `[string]`, em ordem lexicográfica.

Exemplo

No exemplo, os retângulos internos correspondem à saída que seu programa deve gerar:

```
+memorias postumas de bras cubas
+dom casmurro
+melhores poemas
+morte e vida severina
+iracema
$m
```

```
Sugestoes para prefixo "m":
melhores poemas
memorias postumas de bras cubas
morte e vida severina
```

```
$me
```

```
Sugestoes para prefixo "me":
melhores poemas
memorias postumas de bras cubas
```

```
$mem
```

```
Sugestoes para prefixo "mem":
memorias postumas de bras cubas
```

```
$dom
```

```
Sugestoes para prefixo "dom":
dom casmurro
```

```
-melhores poemas
$m
```

```
Sugestoes para prefixo "m":
memorias postumas de bras cubas
morte e vida severina
```

```
#fim
```

Sugestões

- Para comparar duas strings lexicograficamente, você pode usar a função `strcmp`.
- Dado um prefixo $P = c_1 c_2 ... c_{n-1} c_n$, onde c_i corresponde a cada caractere de P representado em ASCII, o conjunto de todas as strings iniciadas por P é dado pelo intervalo $[P, P'[$, onde $P' = c_1 c_2 ... c_{n-1}(c_n + 1)$. Por exemplo, se $P = "aba"$, o intervalo desejado é $["aba", "abb"[$. Note que, devido à representação em ASCII, mesmo que $c_n + 1$ seja um caractere inválido para o problema (`'z' + 1 == '{'`), o intervalo continua sendo válido para obter as strings desejadas.
- Você pode usar o seguinte trecho de código para fazer a leitura:

```
char op;
char arg[80];
/* enquanto lê um caractere e uma string */
while (scanf(" %c %[^\n]", &op, arg) == 2) {
     printf("op='%c'\nst='%s'\n\n", op, arg);
}
```

Observações

- Neste laboratório, é obrigatório implementar e utilizar árvores binárias de busca de acordo com a `struct` que utiliza dois ponteiros para os filhos de um nó, vista em aula.
- **Não** é necessário se preocupar com o balanceamento da árvore.
- Para as turmas A, B e C, esse laboratório tem peso 3.
- Será necessário implementar os arquivos:
  - **`lab08.c`** - Função principal e leitura da entrada
  - **`abb.h`** - Estrutura e protótipo das funções da árvore binária de busca
  - **`abb.c`** - Implementação das funções da árvore binária de busca
- Observações sobre SuSy:
  - Versão do GCC: 4.4.7 20120313 (Red Hat 4.4.7-17)
  - *Flags* de compilação:
    ```
    -g -ansi -pedantic-errors -Wall -Werror -lm
    ```
  - Utilize comentários do tipo `/* comentário */`;
    comentários do tipo `//` serão tratados como erros pelo SuSy
  - Tempo máximo de execução: 2 segundos
- Além das observações acima, esse laboratório será avaliado pelos critérios:
  - Indentação de código
  - Organização
  - Corretude