BCC202 – Estruturas de Dados I

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professora: **Karla A S Joriatti** (`www.decom.ufop.br/`)

Aula Prática II - Recursão

- Procedimento para a entrega:.

1. Submissão: via **Moodle**.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via **Moodle**.
6. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.
7. Siga atentamente quanto ao formato da entrada e saída de seu programa, exemplificados no enunciado.
8. Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.
9. A avaliação considerará o tempo de execução e o percentual de respostas corretas.
10. Eventualmente, serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação.
11. Considere que os dados serão fornecidos pela entrada padrão. Não utilize abertura de arquivos pelo seu programa. Se necessário, utilize o redirecionamento de entrada.
12. Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.
13. Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.
14. Códigos ou funções prontas específicos de algoritmos para solução dos problemas elencados não são aceitos.
15. Não serão considerados algoritmos parcialmente implementados.

- Bom trabalho!

Reconhecedor

Nesta atividade você deverá criar um programa que vai ler uma palavra e dizer se ela pertence ou não a duas linguagens definidas como se segue:

$$L_1 = \{a^n b^n | n \geq 0\} \tag{1}$$

$$L_2 = \{w | w \textit{ tem a mesma quantidade de } a\textit{'s e } b\textit{'s}\} \tag{2}$$

Você deve criar duas funções separadas, *reconheceL1* e *reconheceL2*, que vão receber a palavra, analisá-la recursivamente e retornar se a palavra pertence ou não àquela linguagem. Ambas as linguagens ocorrem sobre o alfabeto $\Sigma = \{a, b\}$.

Especificação da Entrada e da Saída

Seu algoritmo deve ler uma palavra composta por *a's* e *b's*

A saída do programa deve imprimir na tela se a palavra de entrada pertence à linguagem L_1 ou não e se pertence à linguagem L_2 ou não.

Entrada	**Saída**
`aabb`	`Pertence a linguagem L1` `Pertence a linguagem L2`

Entrada	**Saída**
`abab`	`Nao pertence a linguagem L1` `Pertence a linguagem L2`

Entrada	**Saída**
`ababb`	`Nao pertence a linguagem L1` `Nao pertence a linguagem L2`

Siga o protótipo fornecido, sem alterar os nomes dos arquivos e das funções principais e execute sua implementação com os testes disponibilizados, comparando a saída esperada da sua saída.

Diretivas de Compilação

As seguintes diretivas de compilação devem ser usadas (essas são as mesmas usadas no run.codes).

```
$ gcc -c aluno.c -Wall
$ gcc -c pratica.c -Wall
$ gcc aluno.o pratica.o -o exe
```

Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta *valgrind*. Um exemplo de uso é:

```
1 gcc -g -o exe *.c -Wall
2 valgrind --leak-check=full -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
1 ==xxxxx== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.