Departamento de Ciência da Computação – IME-USP

# MAC 115 – Introdução à Computação para Ciências Exatas e Tecnologia

Escola Politécnica – Primeiro Semestre de 1997

Terceiro Exercício-Programa — Data de entrega: até 22 de maio de 1997.
Turmas: Civil B, Mecânica, Naval e Química

**Um formatador de textos simplificadíssimo**

O objetivo deste exercício-programa é construir uma biblioteca de funções para serem utilizadas no desenvolvimento de um formatador de textos. Um **formatador de textos** é um programa cuja entrada consiste de um texto formado por: palavras, espaços em branco, sinais de pontuação, etc. Um texto contém também comandos para o formatador, isto é, indicativos de como o texto deverá ser formatado (parágrafos, linhas centralizadas, identação, etc). As palavras são separadas por um ou mais espaços em branco e a saída consiste do texto de entrada editado como, por exemplo, este que você esta lendo. Os comandos para o formatador, que estão embutidos no texto de entrada, determinam o formato do texto a ser gerado como saída.

Neste exercício-programa você não precisa se preocupar ainda com o comportamento geral de um formatador de textos. Sua tarefa consiste em desenvolver o conjunto de rotinas especificado abaixo e testar o funcionamento de sua implementação. É importante que suas funções sejam desenvolvidas com qualidade, já que você as utilizará no próximo exercício-programa.

## As funções que você deve implementar

Neste exercício-programa, você deve construir e testar as seguintes funções:

**int e_vogal (char s)**
Retorna o valor 1 se *s* for uma vogal, e 0 no caso contrário.

**int num_brancos (char linha[ ], int pos, int tam)**
Retorna o tamanho da seqüência de brancos começando na posição *pos* de vetor *linha* que tem tamanho *tam*. Considere o seguinte exemplo para um vetor *linha* com tamanho *tam* = 35:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | T | e | m | | | | b | a | s | t | a | n | t | e | | | b | r | a | n | c | o | | . | | | |

- num_brancos(linha, 0, tam) = 8
- num_brancos(linha, 10, tam) = 0 (posição 10 não tem um branco e sim um ‘m’)
- num_brancos(linha, 33, tam) = 2

**int tamanho_palavra(char linha[ ], int pos, int tam)**
Esta função tem como entrada:

- linha: vetor que armazena uma seqüência de caracteres
- pos: um índice no vetor
- tam: tamanho do vetor

e devolve o tamanho da palavra que se <u>inicia</u> na posição *pos* de *linha*.
Você pode supor que $0 \leq \text{pos} < \text{tam}$.
Consideraremos que fazem parte de uma palavra letras, dígitos. símbolos de pontuação, etc. Ou seja, o fim de uma palavra é delimitado por um branco (ou pelo final da linha).
Considere o seguinte vetor *linha* de $tam = 33$:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| U | m | | | e | x | . | , | | b | e | m | | b | o | b | o | ! | ! | | E | | , | p | a | r | a | | q | u | e | | ? |

Temos:

- tamanho_palavra(linha, 0, tam) = 2
- tamanho_palavra(linha, 1, tam) = 1 (posição 1 indica palavra começada na letra 'm')
- tamanho_palavra(linha, 3, tam) = 0 (a posição tem um espaço em branco)
- tamanho_palavra(linha, 4, tam) = 4 (tamanho de "ex.," )
- tamanho_palavra(linha, 13, tam) = 6 (tamanho de "bobo!!")
- tamanho_palavra(linha, 22 , tam) = 5 (tamanho de ",para")
- tamanho_palavra(linha, 32, tam) = 1 (tamanho de "?")

**void insere_brancos_iniciais (int n, char linha[ ], int tam)**
Insere $n$ brancos no início do vetor *linha* (deslocando o conteúdo de *linha* para a direita). Exemplo para $tam = 30$, $n = 5$:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| U | m | | e | x | e | m | p | l | o | . | | | | | | | | | | | | | | | | | | | |

resulta em

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | U | m | | e | x | e | m | p | l | o | . | | | | | | | | | | | | | | |

Para $n = 3$ e $tam = 20$ a entrada

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U | m | | e | x | e | m | p | l | o | | b | e | m | | b | o | b | o |

resulta em

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | U | m | | e | x | e | m | p | l | o | | b | e | m | | b |

**void espalha_branco (char linha[ ], int tam)**
Esta função recebe um vetor de caracteres (*linha*) de tamanho *tam* e espalha os brancos que aparecem no final do vetor uniformemente entre as palavras contidas em *linha*. Exemplo para $tam = 25$

Entrada:

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | E | s | t | e | | e | ’ | | u | m | | e | x | e | m | p | l | o | | | | | |

Saída:

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | E | s | t | e | | | | e | ’ | | | | u | m | | | e | x | e | m | p | l | o |

Observe que os brancos no início da linha ficam intactos. Os 5 brancos no final da linha são distribuídos entre as quatro palavras existentes resultando no <u>acréscimo</u> de 2 brancos entre <u>Este</u> e <u>e’</u>, 2 brancos entre <u>e’</u> e <u>um</u>, e apenas 1 branco entre <u>um</u> e <u>exemplo</u>.

**void centraliza (char linha[ ], int tam)**
Esta função recebe um vetor de caracteres (*linha*) de tamanho *tam* e centraliza o texto que aparece no vetor, ou seja, o texto com um mesmo número de brancos iniciais e finais. Exemplo para $tam = 30$:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | U | m | | e | x | e | m | p | l | o | . | | | | | | | | | | | | | | |

resulta em

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | U | m | | e | x | e | m | p | l | o | . | | | | | | | | | |

**FILE* abre_arq_entrada ()**
Esta função deve perguntar ao usuário o nome do arquivo a ser aberto para leitura e devolver uma variável do tipo "arquivo" (FILE *) correspondente a este arquivo. Como no segundo EP (em que um arquivo era aberto para conter a saída da execução), você deve usar a função *fopen*[1]:

[1] O comportamento da função *fopen* é usualmente o mesmo, independente do compilador sendo usado. Mas lembramos que, como para qualquer outra função de biblioteca a ser usada em seu programa, se você não usa o Quick C é sua responsabilidade checar (em geral através do HELP ou manual disponível) que o comportamento na sua plataforma de programação é compatível com o que aqui descrevemos.

FILE* fopen(char filename[ ], "r") abre um arquivo para leitura, enquanto que
FILE* fopen(char filename[ ], "w") abre um arquivo para gravação. A variável *filename* contém o nome do arquivo a ser aberto (por exemplo, "b:dados.txt" para arquivo *dados.txt* do drive B, etc). Em ambos os casos, a função retorna o valor 0 se o arquivo não for encontrado ou não puder ser aberto.

Se o arquivo com o nome fornecido pelo usuário não puder ser aberto, avise-o e peça a entrada de outro nome.

**FILE* abre_arq_saida ()**
Esta função deve perguntar ao usuário o nome do arquivo a ser criado para armazenar a saída do programa. É importante que a sua função verifique se o arquivo com o nome fornecido já existe, neste caso perguntando ao usuário se ele realmente quer que reescrever sobre o arquivo existente.

# Programa Principal

Faça a função *main* que teste cada uma das funções acima. Para isto você pode digitar um arquivo (por exemplo, "b:dados.txt") contendo uma frase[2], e aí durante a execução fornecer o nome do arquivo e tê-lo aberto (função *abre_arquivo*). A frase lida do arquivo deve ser armazenada em um vetor e as várias funções desenvolvidas devem ser chamadas, de forma a testar situações como os vários exemplos descritos neste enunciado. A saída da execução de seu programa deve ser impressa, representando uma lista dos testes que você fez (incluindo valores fornecidos a funções e valores retornados).

Para ler uma linha do arquivo utilize a função fgets:

```
char linha[255];
FILE *arq_entrada;
arq_entrada  = abre_arquivo();
fgets(linha, 255, arq_entrada);
```

**IMPORTANTE:** Todo exercício-programa deve seguir as observações contidas no folheto distribuído juntamente com o primeiro EP. Neste folheto estão descritas as diretrizes para forma de entrega do exercício, aspectos importantes na avaliação, etc.

[2]No próximo EP o arquivo conterá o texto completo a ser manipulado de várias formas.