Exame de Recurso de Programação Imperativa

LCC/MIEF/MIEI

18 de Junho de 2018 – Duração: 2h

Parte A

Considere as seguintes definições de tipos:

```
typedef struct posicao {
    int x, y;
} Posicao;
```

```
typedef struct slist {
   int valor;
   struct slist *prox;
} *LInt;
```

```
typedef struct nodo {
   int valor;
   struct nodo *esq, *dir;
} *ABin;
```

1. Apresente uma definição da função pré-definida em C `char *strstr (char s1[], char s2[])` que determina a posição onde a string `s2` ocorre em `s1`. A função deverá retornar `NULL` caso `s2` não ocorra em `s1`.

2. Defina uma função `void truncW (char t[], int n)` que dado um texto `t` com várias palavras (as palavras estão separadas em `t` por um ou mais espaços) e um inteiro `n`, *trunca* todas as palavras de forma a terem no máximo `n` caracteres. Por exemplo, se a *string* `txt` contiver `"liberdade, igualdade e fraternidade"`, a invocação de `truncW (txt, 4)` deve fazer com que passe a estar lá armazenada a string `"libe igua e frat"`.

3. Defina a função `int maisCentral (Posicao pos[], int N)` que, dado um array com `N` posições, determina o índice da posição que está mais perto da origem (note que as coordenadas de cada ponto são números inteiros).

4. Defina uma função `LInt somasAcL (LInt l)` que, dada uma lista de inteiros, constrói uma nova lista de inteiros contendo as somas acumuladas da lista original (que deverá permanecer inalterada).

   Por exemplo, se a lista `l` tiver os valores `[1,2,3,4]` a lista contruída pela invocação de `somasAcL (l)` deverá conter os valores `[1,3,6,10]`.

5. Apresente uma definição não recursiva da função `int addOrd (ABin *a, int x)` que adiciona um elemento a uma árvore binária de procura. A função deverá retornar `1` se o elemento a inserir já existir na árvore ou `0` no outro caso.

Parte B

Considere o problema de formatar um texto com *justificação* em ambas as margens.
Como input deste problema teremos uma string a ser formatada e o tamanho de cada linha que deve ser produzida.
O resultado é uma sequência de linhas todas com o mesmo comprimento (exceptuando linhas que contenham palavras com tamanho superior ao fixado para cada linha).

Por exemplo, se quisermos formatar o texto `Aqui ao leme sou mais do que eu: Sou um povo que quer o mar que é teu; E mais que o mostrengo, que me a alma teme E roda nas trevas do fim do mundo, Manda a vontade, que me ata ao leme, De El-Rei D. João Segundo!` com uma largura de 30 caracteres, o resultado podia ser o que se apresenta ao lado.

```
Aqui␣ao␣leme␣sou␣mais␣␣do␣␣que
eu:␣Sou␣um␣␣povo␣␣que␣␣quer␣␣o
mar␣que␣e␣teu;␣E␣␣mais␣␣que␣␣o
mostrengo,␣que␣me␣a␣alma␣␣teme
E␣roda␣nas␣trevas␣␣do␣␣fim␣␣do
mundo,␣Manda␣␣a␣␣vontade,␣␣que
me␣ata␣ao␣leme,␣De␣␣El-Rei␣␣D.
Joao␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Segundo!
```

Para resolver este problema vamos utilizar como estrutura de dados auxiliar uma lista em que cada célula contem informação sobre uma palavra do texto, bem como do seu comprimento.

```
typedef struct celula {
    char *palavra;
    int comp;
    struct celula *prox;
} *Palavras;
```

1. Defina uma função `int daPalavra (char *s, int *e)` que calcula o comprimento da primeira palavra dessa string (retorna `0` se não houver palavras). Adicionalmente a função deve colocar no endereço `e` o número de espaços que antecedem essa palavra.

   Use a função `int isspace (char c)` que testa se um caracter é um espaço.

   Por exemplo, a invocação `x=daPalavra(" roda nas trevas", &y)` deve colocar na variável `x` o valor `4` (comprimento de `"roda"`) e na variável `y` o valor `1` (pois há um espaço no início).

2. Usando a função referida na alínea anterior defina agora uma função `Palavras words (char *texto)` que, a partir de um texto, constrói a lista das palavras desse texto.

   A função em questão não precisa de alocar espaço para armazenar cada uma das palavras (o campo `palavra` de cada célula), nem precisa de terminar essas strings com o caracter `'\0'`. Em vez disso deve ser armazenado o endereço onde a palavra se inícia no texto dado, bem como o comprimento da palavra.

3. Defina uma função `Palavras daLinha (Palavras t, int n)` que dada uma lista de palavras `t` e um tamanho de linha `n`, remove dessa lista o maior número de palavras que possam ser escritas numa linha desse comprimento. A função deve deixar em `t` a lista com as palavras que compõe a primeira linha e retornar a lista com apenas as restantes.

   Esta função reorganiza a lista recebida sem alocar memória adicional.

4. Defina uma função `void escreveLinha (Palavras p, int n)` que recebe uma lista de palavras e a escreve no ecran (`stdout`) numa linha com `n` caracteres, correctamente justificada de ambos os lados.

   Assuma que a menos que se trate de uma única palavra com mais do que `n` caracteres, as palavras e os espaços para as separar podem ser escritas com esse número de caracteres. No caso de se tratar de uma linha só com uma palavra maior do que a largura pretendida, deve ser escrita toda a palavra.

5. Usando, entre outras, as funções referidas nas alíneas anteriores defina uma função `void formata (char texto[], int largura)` que **escreve no ecran** o texto formatado com a largura especificada.

   Note que a sua solução deve garantir que é libertada toda a memória alocada.