Algoritmos e Estruturas de Dados I (DCC/003) – 2018/1

Aula Prática 7 – 07/06/2018

Preparem os exercícios de forma que:

1. Os aquivos utilizem a extensão "**.c**"
2. Não seja utilizada função **system**("pause")
3. A função **printf** deve ser utilizada apenas para imprimir a saída do programa.

Atenção: Para esta lista de exercícios é necessário utilizar funções conforme pedem as atividades, caso o contrário, a nota será **0**.

Implemente e execute os exemplos mostrados nos exercícios, verifique os arquivos gerados, faça alterações e então tente resolver as atividades propostas.

1) A manipulação de arquivos permite que uma grande quantidade de dados sejam armazenados e disponibilizados após a execução de um programa. O código abaixo mostra como criar um arquivo texto para armazenar uma *string* de até 100 caracteres lida a partir do teclado:

```
#include <stdio.h>

int main() {
    FILE *fp;
    char str[101];

    // abre o arquivo "arquivo.txt" para escrita, se nao existir cria um novo
    fp = fopen("arquivo.txt", "w");
    if (fp == NULL){
        printf("Erro na abertura do arquivo!\n");
        return 1;
    }

    printf("Digite o texto que sera gravado no arquivo: ");
    // le do teclado (stdin = entrada padrao) no máximo 100 caracteres
    fgets(str, 101, stdin);

    // imprime o valor de str no arquivo representado por fp
    fprintf(fp, "%s", str);
    // fecha o arquivo
    fclose(fp);

    return 0;
}
```

A função **fopen** recebe como parâmetro o nome do arquivo a ser lido/gravado e o modo de abertura do arquivo. No exemplo acima, o arquivo com o nome "arquivo.txt" será criado no diretório de execução do programa. Na linguagem C existe um ponteiro especial do tipo FILE utilizando para manipular arquivos, este ponteiro deve ser usado nas funções de escrita/leitura em arquivos, como a função **fprintf** e **fgets**.

Escreva um programa que leia o nome de um arquivo e em seguida 10 valores inteiros, grave neste arquivo estes valores e a média destes valores. Cada valor deve estar em uma linha do arquivo e a

média ficar na última linha. O programa deve implementar e usar as funções definidas pelos protótipos a seguir:

```
void gravar_vetor(FILE *fp, int v[], int n);
void gravar_media(FILE *fp, int v[], int n);
```

- **Entrada**:
  - o nome do arquivo a ser criado
  - 10 números inteiros.
- **Saída**: **uma** das seguintes mensagens:
  - "Arquivo criado com sucesso!\n"
  - "Erro na abertura do arquivo!\n"

2) A biblioteca *stdio.h* disponibiliza várias funções para leitura e escrita de arquivos texto. O código abaixo mostra exemplos de funções para leitura:

```c
#include <stdio.h>

int main() {
    FILE *fp;
    char str[101], c;

    // abre o arquivo "arquivo.txt" para leitura, se nao existir retorna NULL
    fp = fopen("arquivo.txt", "r");
    if (fp == NULL){
        printf("Erro na abertura do arquivo!\n");
        return 1;
    }

    // le o primeiro caractere presente no arquivo
    c = fgetc(fp);
    printf("Primeiro caractere: %c\n", c);
    // le o segundo caractere presente no arquivo
    c = fgetc(fp);
    printf("Segundo caractere: %c\n", c);

    // volta para o inicio do arquivo
    rewind(fp);

    // le a primeira palavra presente no arquivo
    fscanf(fp, "%s", str);
    printf("Primeira palavra: %s\n", str);

    // volta para o inicio do arquivo
    rewind(fp);

   // le uma string do arquivo de até 100 caracteres ou até que '\n' seja encontrado
    fgets(str, 101, fp);
    printf("Primeira linha: %s\n", str);

    // fecha o arquivo
    fclose(fp);
    return 0;
```

```
}
```

A função **fgetc** retorna um inteiro correspondente a um caractere do arquivo ou o valor -1 (representado pela constante EOF) se a leitura atingiu o final do arquivo. A função **fscanf** é semelhante a função **scanf**, exceto que seu primeiro parâmetro corresponde a um ponteiro para FILE que representa o arquivo a ser lido. A função **rewind** volta a leitura do arquivo para o seu início.

***a*)** Implemente o código acima, verifique o resultado obtido utilizando arquivos gerados pelo exemplo da atividade anterior e gerados manualmente através de um editor de texto simples como o bloco de notas.

***b*)** Apague ou comente os locais que utilizam o comando **rewind** e verifique o que ocorre.

***c*)** Faça um programa para ler o arquivo gerado pelo programa da atividade anterior, os valores deverão ser armazenados em um vetor de 10 inteiros e o desvio padrão destes valores deverá ser impresso na tela.

O programa deve implementar e usar a função definida pelo protótipo abaixo. Esta função deve realizar a leitura do arquivo apontado por ***fp*** e preencher o vetor ***v*** com ***n*** valores.

```
void ler_arquivo(FILE *fp, int v[], int n);
```

- **Entrada**:
  - o nome do arquivo a ser lido
- **Saída**: **uma** das seguintes mensagens:
  - “Desvio padrao igual a %.2f\n”
  - “Erro na abertura do arquivo!\n”

3) Arquivos de texto podem não ser adequados para armazenar valores reais, vetores e outras estruturas de dados mais complexas. Nestes casos o ideal é utilizar arquivos binários. O código a seguir mostra como armazenar o valor de um *double* em um arquivo chamado “dados.bin”:

```
#include <stdio.h>

int main() {
    FILE *fp;
    double v;

    // abre o arquivo "dados.bin" para escrita, se nao existir cria um novo
    fp = fopen("dados.bin", "wb");
    if (fp == NULL){
        printf("Erro na abertura do arquivo!\n");
        return 1;
    }

    printf("Digite um valor real para ser gravado no arquivo: ");
    // le do teclado o valor de v
    scanf("%lf", &v);

    // grava o valor de v no arquivo representado por fp
```

```
    fwrite(&v, sizeof(double), 1, fp);
    // fecha o arquivo
    fclose(fp);

    return 0;
}
```

Observe que o modo de leitura agora tem um 'b' junto com o 'w', isto indica que o arquivo receberá dados binários e não será tratado como um arquivo de texto. A função **fwrite** é responsável por gravar blocos de bytes. Para isto utiliza 4 parâmetros:

1. Endereço de memória onde se encontram os dados a serem gravados no arquivo;
2. Número de bytes por tipo ou unidade do dado, para saber quantos bytes têm o tipo *double* foi utilizado a função **sizeof** com o parâmetro **double**.
3. Número de valores que estão no endereço de memória, no exemplo apenas um valor será lido.
4. Ponteiro para FILE, indicando o arquivo onde os bytes serão escritos

Faça um programa que leia uma matriz de float 3 x 3 e grave estes valores em um arquivo binário. O programa deve implementar e usar a função definida pelo protótipo abaixo:

```
void gravar_matriz(FILE *fp, float m[3][3]);
```

- **Entrada**:
  - o nome do arquivo a ser criado
  - 9 valores reais
- **Saída**: **uma** das seguintes mensagens:
  - "Arquivo criado com sucesso!\n"
  - "Erro na abertura do arquivo!\n"

4) O algoritmo abaixo realiza a leitura do arquivo criado pelo código de exemplo do exercício anterior:

```
#include <stdio.h>

int main() {
    FILE *fp;
    double v;

    // abre o arquivo "dados.bin" para leitura, se nao existir retorna NULL
    fp = fopen("dados.bin", "rb");
    if (fp == NULL){
        printf("Erro na abertura do arquivo!\n");
        return 1;
    }

   // le (sizeof(double) * 1) bytes do arquivo apontado por fp e armazena estes
dados em &v
    fread(&v, sizeof(double), 1, fp);
    printf("Valor real gravado no arquivo: %lf", v);

    // fecha o arquivo
    fclose(fp);
```

```
    return 0;
}
```

Observe que o modo de leitura do arquivo agora é “rb” e que a função **fread** utiliza os mesmos parâmetros da função **fwrite**, mas agora os dados são lidos do arquivo para a memória.

Faça um programa que leia o arquivo gerado pelo programa do exercício anterior e imprima na tela o somatório de cada coluna da matriz. O programa deve implementar e usar a função definida pelo protótipo abaixo:

```
void ler_matriz(FILE *fp, float m[3][3]);
```

Esta função deve preencher a matriz ***m*** a partir do arquivo apontado por ***fp***.

- **Entrada**:
  - o nome do arquivo a ser lido
- **Saída**: **uma** das seguintes mensagens:
  - “%.2f\n” para o somatório de cada coluna, ou
  - “`Erro na abertura do arquivo!\n`”

5) Faça um programa que leia um arquivo informado pelo usuário e uma palavra, depois imprima quantas vezes esta palavra aparece no arquivo. Letras maiúsculas devem ser consideradas diferentes de letras minúsculas. O programa deve implementar e usar a função definida pelo protótipo abaixo:

```
int contar_ocorrencias(char *palavra, FILE *fp);
```

Dica: a função **fscanf** retorna a constante EOF quando atinge o final do arquivo.

- **Entrada**:
  - o nome do arquivo a ser lido
  - uma palavra qualquer
- **Saída**: **uma** das seguintes mensagens:
  - “A palavra %s aparece %i vezes no arquivo %s\n”
  - “`Erro na abertura do arquivo!\n`”