# Algoritmos e Programação I
# Exercício-Programa 4

Marcelo Hashimoto

12 de maio de 2013

## 1 Introdução

Neste Exercício-Programa, seu objetivo é calcular algumas estatísticas simples sobre uma lista de notas de alunos. Porém, desta vez, tanto a entrada como a saída dos dados será através de *arquivos* em vez do terminal.

## 2 Abrindo e fechando um arquivo para leitura

Salve todos os arquivos anexados a este enunciado na mesma pasta e abra o código-fonte `exemplo1.c`.

```
#include <stdio.h>

#define FALSE 0
#define TRUE  1

#define TAM 51

int main() {
  char *s, linha[TAM];
  FILE *entrada;

  entrada = fopen("exemplo1.txt", "r");

  if(entrada == NULL) {
    printf("ERRO: caminho invalido\n");
    return 0;
  }

  while(TRUE) {
    s = fgets(linha, TAM, entrada);

    if(s == NULL)
      break;

    printf("%s\n", linha);
  }

  fclose(entrada);

  return 0;
}
```

Compile e execute esse código-fonte. Nas subseções a seguir ele é explicado em detalhes.

### 2.1 Fluxo de arquivo

A variável `entrada`, um apontador para `FILE`, é um novo exemplo do conceito de *fluxo de dados* visto em aula. O único outro exemplo que havia sido visto até agora era o `stdin`, que representa o teclado.

Essa variável recebe um endereço válido da função `fopen`, que por sua vez recebe como parâmetros duas strings: a primeira é o *caminho* onde o arquivo desejado se localiza e a segunda é o *modo de abertura*. Nesse exemplo em particular temos `"r"`, pois deseja-se abrir o arquivo para *leitura* (*read*).

Nem sempre a abertura é bem-sucedida: o arquivo pode, por exemplo, não existir ou não permitir leitura. Nesse caso, a função devolve o endereço especial `NULL`. Comparando a resposta com `NULL` depois da execução de `fopen`, o programa decide se deve continuar ou terminar. No segundo caso, imprime uma mensagem de erro.

### 2.2 Usando `fgets` em arquivos

Repare agora no uso de `fgets` nesse código-fonte e você vai notar duas diferenças em relação à maneira como você utilizou essa função até agora. A primeira está na passagem de `entrada` como terceiro parâmetro em vez de `stdin`. Isso significa que a função lê uma linha do arquivo em vez do teclado.

A segunda está na atribuição do valor devolvido pela função a uma variável `s`. Esse valor é um endereço de `char`, portanto essa variável é um apontador para `char`. Se esse endereço for igual a `NULL`, a leitura não foi bem-sucedida. Podemos utilizar essa comparação para saber quando as linhas do arquivo terminaram.

Repare também que o programa imprime cinco linhas, embora o arquivo possua quatro. Isso acontece porque na terceira linha o número de caracteres ultrapassa o limite estabelecido pelo segundo parâmetro de `fgets`. Portanto, a primeira chamada consome apenas a primeira metade dela.

### 2.3 Encerrando o fluxo

Quando não há mais nenhuma leitura a ser feita, o arquivo deve ser fechado através da função `fclose`, que recebe o respectivo endereço de `FILE` como parâmetro. Isso libera os recursos alocados durante a abertura.

## 3 Abrindo e fechando um arquivo para escrita

Agora abra, compile e execute o código-fonte `exemplo2.c`.

```
#include <stdio.h>

int main() {
  FILE *saida;

  saida = fopen("exemplo2.txt", "w");

  if(saida == NULL) {
    printf("ERRO: caminho invalido\n");
    return 0;
  }

  fprintf(saida, "Hello World!\n");
  fprintf(saida, "inteiro: %d\n", 1);
  fprintf(saida, "decimal: %f\n", 0.1);
  fprintf(saida, "caractere: %c\n", 'a');

  fclose(saida);

  return 0;
}
```

Nesse código-fonte, temos novamente o uso de `fopen` e `fclose`, mas o primeiro dessa vez recebe o modo de abertura `"w"`, pois deseja-se abrir para *escrita* (*write*). Esse modo cria o arquivo quando ele não existe e **apaga seu conteúdo quando existe, portanto tome muito cuidado para não abrir dessa maneira um arquivo cujo conteúdo é importante!**

Temos também a função `fprintf`, que se comporta como `printf` mas imprime em arquivo em vez da tela.

# 4 Lendo inteiros de um arquivo

Analogamente a `fprintf`, temos naturalmente `fscanf`. Abra o código-fonte `exemplo3.c`.

```
#include <stdio.h>

#define TAM 51

int main() {
  char s[TAM];
  int i;
  FILE *entrada;

  entrada = fopen("exemplo3.txt", "r");

  if(entrada == NULL) {
    printf("ERRO: caminho invalido\n");
    return 0;
  }

  fscanf(entrada, "%d", &i);

  printf("%d\n", i);

  fgets(s, TAM, entrada);

  printf("%s\n", s);

  fclose(entrada);

  return 0;
}
```

Esse programa executa um `fscanf` com `%d` e um `fgets`, ou seja, lê um inteiro e depois uma linha de texto. Portanto, se o conteúdo do arquivo `exemplo3.txt` for

```
10
texto
```

parece natural esperar que ele imprima esse inteiro e essa linha de texto. Porém, se você compilar e executar vai notar que ele não funciona corretamente. Por quê? A resposta e a solução estão ambos no Exercício-Programa 3.

# 5 Especificação

Seu programa deve executar as seguintes operações.

1. Recebe duas linhas de texto do usuário através de duas chamadas da função `fgets`. Essas linhas de texto representam os caminhos de um arquivo de entrada e um arquivo de saída. Você pode supor que ambos possuem no máximo 100 caracteres.
2. Lê o arquivo de entrada. Espera-se que tenha o seguinte formato:
   - um inteiro $n \leq 25$, representando uma quantidade de alunos, seguido por uma quebra de linha;
   - $n$ pares de linhas de texto: a primeira contém um decimal, representando uma nota, e a segunda contém um nome que possui no máximo 50 caracteres.
3. Escreve o arquivo de saída. Espera-se que tenha as seguintes informações:
   - média dos alunos;
   - lista dos alunos acima da média;

- nome do melhor aluno.

4. Termina.

Um exemplo de entrada segue abaixo. O formato da saída é livre.

```
4
5.0
Tom Araya
2.0
Jeff Hanneman
9.0
Kerry King
10.0
Dave Lombardo
```

Espera-se que seu programa mostre mensagens de erro adequadas quando o arquivo não pode ser aberto, o formato está incorreto ou dados estão faltando. Mas não se preocupe com dados sobrando.

Note que você vai precisar armazenar todos os nomes. Isso pode ser feito usando uma *matriz de caracteres*.

# 6 Detalhes de entrega

- Este exercício deve ser feito individualmente ou em grupo de 2 a 3 alunos.
- O programa deve ser entregue pelo Blackboard até as **23:50** do dia **1 de junho**.
- Entregue apenas um arquivo de código-fonte, como anexo e com extensão `c`.

O enunciado foi projetado para um prazo menor do que o estabelecido acima. Você pode entregar em cima da hora, mas **problemas técnicos do Blackboard não serão aceitos como justificativa para atrasos.**

# 7 Critérios de correção

- **Grupo com mais de** 3 **alunos:** nota zero.
- **Entrega atrasada:** nota zero.
- **Arquivos a mais:** nota zero.
- **Entrega não é anexo:** nota zero.
- **Entrega não possui extensão `c`:** nota zero.
- **Erro durante compilação:** nota zero.
- **Aviso durante compilação:** desconto de `1.0` ponto por aviso.
- **Uso de conceito que não foi ensinado:** nota zero.

Se qualquer tipo de plágio for constatado, **todos** os membros de **todos** os grupos envolvidos **no mínimo** receberão nota zero. Outras punições adicionais ainda poderão ser estabelecidas posteriormente pela coordenação.