Aula 26

MC 102 - Algoritmos e Programação de Computadores

Arquivos.

Armazenamento

Todos os programas que fizemos até agora não guardavam os dados calculados de forma definitiva, nem liam dados salvos em arquivos para agilizar o seu processamento: os resultados deviam ser anotados e os dados de entrada deviam ser todos digitados.

Armazenamento

Os arquivos são úteis para permitir registrar as informações no computador, para reúso futuro. Dados que seriam digitados constantemente podem também ser armazenados em arquivos e posteriormente ser utilizados pelos programas.

Arquivos

Para utilizarmos arquivos de dados, temos que ter em mente as principais operações em um arquivo:

Abrir / Ler / Escrever / Fechar

Características de arquivos:

- Armazenam grande quantidade de informação
- Persistência dos dados
  - Disco rígido, pendrive, CD, DVD, etc..
- Concorrência do acesso à informação
  - Mais de um usuário pode utilizar o mesmo arquivo
- Acesso não seqüêncial aos dados
  - Acesso a uma parte específica do arquivo

Mais sobre Arquivos

Todo arquivo deve possuir necessariamente um **nome**. Seu nome pode conter uma **extensão** identificando seu conteúdo. Isso não é obrigatório, apesar de recomendado.

Nome	Tipo de arquivo
lista.txt	arquivo texto simples
prog.c	código fonte C (arquivo texto)
trabalho.doc	documento MSWord
programa.exe	Executável (Windows)
lab*	Executável (Linux)
arq.pdf	*Portable Document Format*

Tipos de Arquivos

Arquivos podem ter o mais variado conteúdo, mas do ponto de vista dos programas existem apenas dois tipos de arquivo:

- Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código fonte C, documento texto simples, páginas HTML.
- Arquivo binário: Seqüência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente. Exemplos: arquivos executáveis, arquivos compactados, documentos do Word

Organização de Arquivos

Um grande volume de dados obriga-nos a manter uma certa “ordem” no armazenamento dos arquivos. De nada adiantaria guardar os dados se não conseguíssemos encontrá-los facilmente.

Dessa forma, somos quase que obrigados a utilizar **Pastas** ou **Diretórios**, que representam estruturas especiais no sistema de arquivos para o armazenamento de “grupos” de arquivos de forma organizada.

Uma pasta pode conter mais subpastas, e assim sucessivamente. Porém, um arquivo deve ser único (somente uma cópia) em cada pasta.

Localização de Arquivos

Há uma diferença na representação da localização dos arquivos no Windows e no Linux. Vejamos um exemplo:

Windows: C:\MC102\lab09**lab09.c**

Linux: /home/usuario/mc102/lab09/**lab09.c**

A localização de um arquivo é denominada **Caminho**. Se todo o caminho é informado, temos um *caminho completo*. Se o caminho é considerado a partir do diretório corrente (sem o '/' ou o “c:\”), temos um *caminho relativo*.

Caminhos

Caminho Completo (ou absoluto)

Windows: C:\MC102\lab09**lab09.c**

Linux: /home/usuario/mc102/lab09/**lab09.c**

Caminho Relativo

Se o programa estiver sendo executado na pasta C:\MC102 no Windows ou no linux no diretório /home/usuario/mc102/, podemos fazer referência apenas ao seu caminho relativo:

Windows: lab09**lab09.c**

Linux: lab09/**lab09.c**

Permissões dos Arquivos

No Windows, as permissões são mais complicadas e, em geral, todos os arquivos podem ser acessados pelos usuários (exceto aqueles protegidos pelos administradores). No Linux, basicamente temos as sequintes permissões (use `ls -l` para listar os arquivos com permissões):

Permissão	Significado
r	Leitura permitida
w	Escrita permitida
x	Execução (arquivos) ou entrada (para diretórios) permitida

```
$ ls -l
-rw-r----- 1 fulano alunos  545 Dez  8  2006 prog.c
drwxr-xr-x 2 fulano alunos 4096 Fev 22 01:14 mc102/
```

diretório | permissões | usuario | grupo | tamanho | data/hora | nome

Trabalhando com Arquivos em C

Para trabalharmos com arquivos na linguagem C, devemos usar a biblioteca de funções **`stdlib.h`**. É por meio desta biblioteca que temos acesso à estrutura FILE que armazena na memória informações básicas sobre o arquivo, além das principais funções de manipulação de arquivos que veremos mais adiante.

Declarando uma variável como arquivo texto:

```
FILE *meu_arquivo;
```

Obs: Note que o tipo FILE deve obrigatoriamente estar em maiúsculas!

Abrindo Arquivos

Abrir um arquivo texto:

```
FILE * fopen(char *nome_arq, char *modo)
```

aloca espaço para a estrutura FILE, atribui informações do arquivo para ela e retorna o endereço desta estrutura na memória.

Parâmetros:

nome_arq: String que contém o caminho (completo ou relativo) e o nome do arquivo que será aberto

modo: String que define o modo de abertura do arquivo, que pode ser:

r	somente para leitura (**o arquivo deve existir**)
w	somente para escrita a partir do início do arquivo (se o arquivo não existir, cria um novo. Se já existir, apaga seu conteúdo antes da escrita)
a	somente escrita no final de um arquivo (se o arquivo não existir, cria um novo)
r+	leitura e escrita no início do arquivo (**o arquivo deve existir**)
w+	leitura e escrita no início do arquivo (apaga o conteúdo se o arquivo existir ou cria um novo arquivo caso contrário)
a+	abre para leitura no início do arquivo e escrita no seu final

Abrindo Arquivos

Exemplo:

```
FILE *arq;
arq = fopen("teste.txt", "r");
```

Como saber se o arquivo foi relmente aberto?

O resultado de fopen pode ser um apontador nulo (NULL). Neste caso, basta verificar se (farq != NULL).

```
FILE *arq;
arq = fopen("teste.txt", "r");
if (arq == NULL) {
  printf("Erro ao abrir aquivo.\n");
  return 0;
}
```


Escrevendo em um Arquivo

Escreve dados em um arquivo texto:

```
int fprintf(FILE *arq, char *fmt, ...)
```

Parâmetros:

arq: Apontador para a estrutura arquivo (FILE)

fmt: Formato de escrita (o mesmo usado em printf)

...: Variáveis ou constantes com os valores a serem gravados

Exemplo:

```
FILE *arq; int i=10, double n=7.5;
arq = fopen("teste.txt", "r");
if (arq != NULL)
  fprintf(arq, "%d %lf %s\n", i, n, "1a. Linha");
...
```


Lendo Dados de um Arquivo

Lê dados de um arquivo texto:

```
int fscanf(FILE *arq, char *fmt, ...)
```

Parâmetros:

arq: Apontador para a estrutura arquivo (FILE)

fmt: Formato de leitura (o mesmo usado em scanf)

...: Endereço das variáveis nas quais serão atribuídos os valores lidos do arquivo

Retorno:

EOF quando atinge o final do arquivo

Exemplo:

```
FILE *arq; int i, double n;
arq = fopen("teste.txt", "r");
if (arq != NULL)
  fscanf(arq, "%d %lf", &i, &n);
...
```


Lendo Dados de um Arquivo

Lê uma string (uma linha) de um arquivo:

```
int fgets(char *str, int tam, FILE *arq)
```

Parâmetros:

str: String na qual será armazenada a linha lida.

tam: Tamanho máximo da string str.

arq: Apontador para a estrutura arquivo (FILE)

Retorno:

str em caso de sucesso na leitura

NULL quando atinge o final do arquivo

Exemplo:

```
FILE *arq; char s[101];
arq = fopen("teste.txt", "r");
if (arq != NULL) {
  fgets(s, 100, arq);
  printf("%s", s);
}
```

Fechando um Arquivo

Fecha um arquivo aberto e libera a estrutura da memória:

```
int fclose(FILE *arq)
```

Parâmetros:

arq: Apontador para a estrutura arquivo (FILE)

Exemplo:

```
FILE *arq; char s[101];

arq = fopen("teste.txt", "r");
if (arq != NULL) {
  fgets(s, 100, arq);
  printf("%s", s);
}
fclose(arq);
```


Exemplo 1

Ler um vetor de n inteiros do teclado e gravar em um arquivo

```
int main(int argc, char **argv) {
  FILE *arq; int i, n, *v;

  printf("Informe o numero de elementos: ");
  scanf("%d", &n);
  v = (int *) malloc (n * sizeof(int));
  printf("Informe os elementos:\n");
  for(i=0; i<n; i++)
    scanf("%d", v + i);

  arq = fopen("vetor.txt", "w");
  if (arq != NULL) {
    fprintf(arq, "%d\n", n);
    for(i=0; i<n; i++)
      fprintf(arq, "%d ", v[i]);
    fclose(arq);
  }
  free(v);
  return 0;
}
```


Exemplo 2

Ler um vetor de n inteiros de um arquivo e imprimir na tela.

```
int main(int argc, char **argv) {
  FILE *arq;
  int i, n, *v;

  arq = fopen("vetor.txt", "r");
  if (arq == NULL)
    return 0;

  fscanf(arq, "%d", &n);
  v = (int *) malloc (n * sizeof(int));
  for(i=0; i<n; i++)
    fscanf(arq, "%d", v + i);

  printf("Vetor de %d inteiros\n", n);
  for(i=0; i<n; i++)
    printf("%d ", v[i]);

  printf("\n");
  fclose(arq);
  free(v);
}
```


Exercício 1

Escreva um programa que leia um conjunto de inteiros de um arquivo, calcule a sua média e grave este valor no final do mesmo arquivo.