MC-102 — Aula 21
Arquivos

Instituto de Computação – Unicamp

Segundo Semestre de 2009

Roteiro

Arquivos

Características

- Podem armazenar grande quantidade de informação.
- Dados são persistentes (gravados em disco).
- Acessso aos dados pode ser não sequencial (acesso direto a registros em um banco de dados).
- Acesso à informação pode ser concorrente (mais de um programa ao mesmo tempo).

Nomes e extensões

- Arquivos são identificados por um nome.
- O nome de um arquivo pode conter uma extensão que indica o conteúdo do arquivo.

Algumas extensões

arq.txt	arquivo texto simples
arq.c	código fonte em C
arq.pdf	*portable document format*
arq.html	arquivo para páginas WWW *(hypertext markup language)*
arq*	arquivo executável (UNIX)

Tipos de arquivos

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

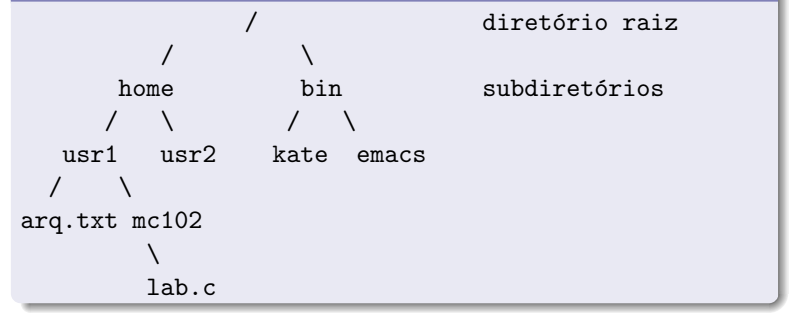
Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código fonte C, documento texto simples, páginas HTML.

Arquivo binário: Sequência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente. Exemplos: arquivos executáveis, arquivos compactados, documentos do Word.

Diretório

- Também chamado de pasta.
- Contém arquivos e/ou outros diretórios.

Uma hierarquia de diretórios



Caminhos absolutos ou relativos

O nome de um arquivo pode conter o seu diretório, ou seja, o caminho para encontrar este arquivo a partir da raiz. Os caminhos podem ser especificados de duas formas:

Caminho absoluto: descrição de um caminho desde o diretório raiz.

```
/bin/emacs
/home/usr1/arq.txt
```

Caminho relativo: descrição de um caminho desde o diretório corrente.

```
arq.txt
mc102/lab.c
```

Para ver qual é o diretório corrente, use o comando `pwd`. Para mudar de diretório, use o comando `cd`.

Atributos de arquivos

Além do nome, arquivos possuem vários outros atributos:

- Nome do arquivo
- Proprietário do arquivo
- Datas de criação, alteração e acesso
- Tamanho em bytes
- Permissão de acesso

Para ver estes atributos, use os comandos `ls -l`.

Permissão de acesso

Existem três níveis de controle: proprietário, grupo e todos.

```
$ ls -l
-rw-r----- 1 jose alunos  545 Nov  8  2005 cp.c
drwxr-xr-x 2 jose alunos 4096 Jun  6 14:54 mc102/
```

- r: leitura
- w: escrita
- x: execução para arquivos, permissão de entrada para diretórios

Abrindo um arquivo para leitura

- Antes de acessar um arquivo, devemos abri-lo com a função `fopen()`.
- Em caso de erro a função retorna `NULL`.
- A função `perror()` obtém e exibe uma mensagem explicativa.

Abrindo o arquivo `teste.txt`

```
if (fopen("teste.txt", "r") == NULL)
  perror("Erro ao abrir o arquivo.\n");
else
  printf("Arquivo aberto para leitura.\n");
```

Veja o exemplo em `fopen-r.c`.

Lendo dados de um arquivo

- Não basta chamar a função `fopen()`, temos que pegar o seu valor de retorno (um apontador para *stream*).
- Para ler dados do arquivo, usamos a função `fscanf()`, semelhante à função `scanf()`.
- Para fechar o arquivo usamos a função `fclose()`.

Lendo dados do arquivo `teste.txt`

```
FILE *f = fopen ("teste.txt", "r");
while (fscanf(f, "%c", &c) != EOF)
  printf("%c", c);
fclose(f);
```

Veja o exemplo em `fscanf.c`.

Escrevendo dados em um arquivo

- Para escrever em um arquivo, ele deve ser aberto de forma apropriada.
- Usamos a função `fprintf()`, semelhante a função `printf()`.

Copiando dois arquivos

```
FILE *fr = fopen ("teste.txt", "r");
FILE *fw = fopen ("saida.txt", "w");
while (fscanf(fr, "%c", &c) != EOF)
  fprintf(fw,"%c", c);
fclose(fr);
fclose(fw);
```

Veja o exemplo em `fprintf.c`.

fopen

Um pouco mais sobre a função `fopen()`.

```
FILE* fopen(const char *caminho, char *modo);
```

Modos de abertura de arquivo

modo	operações	ponto no arquivo
r	leitura	início
r+	leitura e escrita	início
w	escrita	início
w+	leitura e escrita	início
a	escrita	final
a+	leitura	início
	escrita	final

Lendo um vetor de um arquivo

```
FILE  *fr;
int i, n, *v;

fr = fopen ("v-in.txt", "r");
fscanf(fr, "%d", &n);  /* Dimensão do vetor */
v = (int *) malloc (n * sizeof(int));
for (i = 0; i < n; i++)
  fscanf(fr, "%d", &v[i]);
fclose(fr);
```

Veja o exemplo em `le_vetor.c`.

Escrevendo um vetor em um arquivo

```
FILE  *fw = fopen ("v-out.txt", "w");
fprintf(fw, "%d\n", n); /* Dimensão do vetor */
for (i = 0; i < n; i++)
  fprintf(fw, "%d\n", v[i]);
fclose(fw);
```

Veja o exemplo em `le_vetor.c`.

Lendo uma matriz de um arquivo

- Para usar alocação dinâmica, uma solução é criar um vetor linear de dimensão `nlin * ncol`.

```
int *v = (int *)
         malloc (nlin * ncol * sizeof(int));
for (i = 0; i < nlin * ncol; i++)
  fscanf(fr, "%d", &v[i]);
```

Veja o exemplo em `le_matriz.c`.

Escrevendo uma matriz em um arquivo

- Para gravar uma matriz, usamos a idéia inversa, ou seja: `mat[i][j] = v[i*ncol + j]`.

```
for (i = 0; i < nlin; i++)
  for (j = 0; j < ncol; j++)
   fprintf(fw, "mat[%d][%d] = %d\n",
           i, j, v[i*ncol + j]);
```

Veja o exemplo em `le_matriz.c`.

Lendo uma matriz de um arquivo

- Uma outra forma de fazer alocação dinâmica consiste em criar `nlin` vetores de `ncol` inteiros.

```
int **v = (int **) malloc(nlin * sizeof(int*));
for (i = 0; i < nlin; i++)
  v[i] = (int *) malloc(ncol * sizeof(int));
for (i = 0; i < nlin; i++)
  for (j = 0; j < ncol; j++)
    fscanf(fr, "%d", &v[i][j]);
```

Veja o exemplo em `le_matriz2.c`.

Argumentos para o main

- Como já vimos, o bloco `main` é uma função.
- Esta função recebe argumentos da linha de comando.

```
int main (int argc, char* argv[]) {
  FILE *fr, *fw
  if (argc < 3) {
    printf("Uso: %s <origem> <destino>\n",
            argv[0]);
    return 1;
  }
  fr = fopen (argv[1], "r");
  fw = fopen (argv[2], "w");
```

Veja o exemplo em `cp.c`.