Introdução à Programação
Arquivos

Prof. Tiago A. Almeida

talmeida@ufscar.br

Departamento de Computação

Universidade Federal de São Carlos

Introdução à Programação

Arquivos

- Podem armazenar grande quantidade de informação
- Dados são persistentes (gravados em disco)
- Acesso 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)

Introdução à Programação

Nomes e extensões

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

arq.txt	Arquivo texto simples
arq.c	Código-fonte em C
arq.pdf	*Portable document format*
arq.html	Arquivo para páginas WWW
...	...
arq.exe	Arquivo executável (Windows)

Introdução à Programação

Tipos

- 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

Introdução à Programação

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

- Para ver qual é o diretório corrente, use o comando `pwd`

Introdução à Programação

Declaração

- Em C, arquivos são manipulados como variáveis do tipo apontador para arquivo, as quais são declaradas da seguinte forma:

```
int main () {
   FILE *aparq;
   …
   return 0;
}
```

- As funções necessárias para manipulação estão na `stdio.h`

Introdução à Programação

Abertura e fechamento

- Antes de acessar os dados dentro de um arquivo é necessário abri-lo (`fopen`)
- Ao terminar de realizar as operações no arquivo, devemos fechá-lo (`fclose`)
- As operações de abertura e fechamento garantem a integridade dos dados

Introdução à Programação

Abertura

- `FILE* fopen(char *nome, char *modo);`
  - Abre o arquivo **nome** com o **modo** dado
  - **nome**: string que pode incluir um caminho completo. O separador de diretórios depende do sistema operacional
    - No Windows: `"C:\\temp\\arquivo"` é um nome válido
    - No Linux: `"/usr/arquivo"` é um nome válido
  - **modo**: string que indica se o arquivo será aberto para leitura, escrita, ambos, ou variações
    - Modo `"r"` – apenas leitura (início). O arquivo deve existir.
    - Modo `"w"` – apenas escrita (início). Cria um novo arquivo.
    - Modo `"a"` – apenas escrita, mantém os dados originais (final). Cria arquivo caso não exista.
    - Modo `"r+"` – leitura e escrita (início). O arquivo deve existir.
    - Modo `"w+"` – leitura e escrita (início). Cria um novo arquivo.
    - Modo `"a+"` – leitura (início) e escrita (final). Cria arquivo caso não exista.

Introdução à Programação

Abertura

- `FILE* fopen(char *nome, char *modo);`
  - Se a abertura falhar, `fopen` retora `NULL`
  - A função `perror()` obtém e exibe uma mensagem explicativa

Introdução à Programação

Fechamento

- `int fclose(FILE *f);`
  - Fecha um arquivo aberto. Retorna 0 em caso de sucesso, outro valor em caso de erro (como tentar fechar um arquivo que não está aberto)

Introdução à Programação

Abertura e fechamento

- Ver `fopen-r.c`

```
FILE *aparq; //declaração

//Abre o arquivo em modo leitura
aparq = fopen("teste.txt", "r");

if (aparq == NULL) // Testa a abertura
   perror("Erro ao abrir o arquivo.\n");
else
   printf("Arquivo aberto para leitura.\n");

fclose(aparq); // Libera o apontador
```

Introdução à Programação

Leitura de dados

- C provê funções para ler caracteres, linhas de texto, padrões formatados ou vetores de bytes:

- `int fgetc(FILE *f)`
  - lê um byte, retorna seu valor (0255) como inteiro. Retorna EOF em caso de erro

- `char* fgets(char *dest, int limite, FILE *f);`
  - lê uma linha de texto (ou até atingir o limite de bytes lidos) e guarda o texto lido em **`dest`**. Retorna **`dest`** em caso de sucesso, **`NULL`** em caso de erro

- `int fscanf(FILE *f, char *formato, ...);`
  - `scanf` para arquivos

Introdução à Programação

Leitura de dados - exemplo

- Ver `fscanf.c`

```
FILE *f;
int num;

f = fopen ("teste.txt", "r");

if (f == NULL) {
   perror("teste.txt");
   return 1;
}

while (fscanf(f, "%d", &num) != EOF)
   printf("%d ", num);

fclose(f);
```

Introdução à Programação

Escrita de dados

- C provê funções para escrever caracteres, strings, padrões formatados ou vetores de bytes:

- `int fputc(int c, FILE *f);`
  - Escreve o byte `c`. Retorna EOF em caso de erro

- `int fputs(char *dest, FILE *f);`
  - Escreve a string `dest` (que deve ter o zero terminador) no arquivo. Retorna `EOF` em caso de erro

- `int fprintf(FILE *f, char *formato, ...);`
  - `printf` para arquivos

Introdução à Programação

Escrita de dados - exemplo

- Ver `fprintf.c`

```
FILE  *fr, *fw;
int num;

fr = fopen ("teste.txt", "r");
fw = fopen ("saida.txt", "w");

if (fr == NULL){
    perror("teste.txt"); exit(EXIT_FAILURE);
}

if (fw == NULL){
    perror("saida.txt"); exit(EXIT_FAILURE);
}

while (fscanf(fr, "%d", &num) != EOF)
    fprintf(fw, "%d ", num);

fclose(fr);
fclose(fw);
```

Introdução à Programação

Outras funções

- `int remove(const char *filename)`
  - Remove o arquivo com nome informado. Retorna 0 em caso de sucesso
    - Ex: `remove("arquivo.txt");`

- `int rename(const char *old, const char *new)`
  - Renomeia arquivo. Retorna 0 em caso de sucesso
    - Ex: `rename("arq1.txt","arq2.txt");`

- `void rewind(FILE *f)`
  - Reposiciona o "indicador" no início do arquivo
    - Ex: `rewind(f);`

- Consulte: `http://www.cplusplus.com/reference/cstdio/`

Introdução à Programação

Ler vetor de um arquivo

- Ver `le_vetor.c`

```
FILE *fr;
int i, n, v[100];

fr = fopen ("v-in.txt", "r");

fscanf(fr, "%d", &n); /* Lê a dimensão do vetor */

for (i = 0; i < n; i++)
   fscanf(fr, "%d", &v[i]);

fclose(fr);
```

Introdução à Programação

Escrever vetor em um arquivo

- Ver `le_vetor.c`

```
FILE *fw = fopen ("v-out.txt", "w");

fprintf(fw, "%d\n", n); /* Escreve a dimensão do vetor */

for (i = 0; i < n; i++)
   fprintf(fw, "%d\n", v[i]);

fclose(fw);
```

Introdução à Programação

Ler matriz de um arquivo

- Ver `le_matriz.c`

```
FILE *fr;
int i, j, nlin, ncol, m[100][100];

fr = fopen ("m-in.txt", "r");

fscanf(fr, "%d %d", &nlin, &ncol);

for (i = 0; i < nlin; i++)
   for (j = 0; j < ncol; j++)
      fscanf(fr, "%d", &m[i][j]);

fclose(fr);
```

Introdução à Programação

Escrever uma matriz em um arquivo

- Ver `le_matriz.c`

```
FILE *fw;

fw = fopen ("m-out.txt", "w");

/* Escreve as dimensoes da matriz */
fprintf(fw, "%d %d\n", nlin, ncol);

for (i = 0; i < nlin; i++) {
   for (j = 0; j < ncol; j++)
      fprintf(fw, "%d ", m[i][j]);
   fprintf(fw, "\n");
}

fclose(fw);
```

Introdução à Programação

Ler e escrever registros em um arquivo

- Campos e registros são geralmente separados por delimitadores definidos no programa
- O programa deverá ser capaz de escrever e ler os campos registros seguindo regras pré-definidas

- Ver `registros.c`