ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES

Disciplina: 116301

Profa. Carla Denise Castanho

Universidade de Brasília – UnB
Instituto de Ciências Exatas – IE
Departamento de Ciência da Computação – CIC

12. ARQUIVOS BINÁRIOS

Arquivos

- Como você deve se lembrar, um arquivo é uma **abstração** do SO para lidar com dados **não voláteis**.
- Do ponto de vista do programador, um arquivo é simplesmente uma *stream*, *i.e.*, uma **sequência de *bytes***.
- Nós já estudamos os arquivos tipo **texto**, em que os *bytes* são interpretados como caracteres.
- Nesse capítulo, estudaremos arquivos em formato **binário**, que é o mais genérico e de mais baixo nível.

Arquivos Binários

- Quando trabalhamos com arquivos binários, nós realizamos leituras e escritas de **blocos de um ou mais *bytes***.
- A vantagem desse formato é que podemos guardar **estruturas** inteiras, como **registros** ou **vetores**.
- Além disso, podemos ler e escrever os dados **diretamente no formato em que estão na memória**. Por exemplo, uma variável inteira ou real.
- A desvantagem é que os dados não são fáceis de serem **entendidos por um ser humano**!
- Para ver o conteúdo de um arquivo texto, por exemplo, podemos simplesmente **abri-lo em um editor**; com um arquivo binário, temos que usar um programa especial, que nos mostre o **valor de cada *byte*** (normalmente em hexadecimal), e **interpretá-los manualmente, um por um**.

Abrindo Arquivos Binários

- Para abrir um arquivo binário, nós também utilizamos a função *fopen*, porém, os códigos das operações passam a ter o sufixo “b”:
  - **rb** (*read*) - abre um arquivo BINÁRIO para leitura, ele já deve existir;
  - **wb** (*write*) - abre (ou cria) um arquivo BINÁRIO para gravação, se já existir, elimina seu conteúdo e recomeça a gravação a partir do seu início;
  - **ab** (*append*) - abre (ou cria) um arquivo BINÁRIO para gravação, e sempre grava a partir de seu final;
  - **r+b** - abre um arquivo BINÁRIO para leitura e gravação; o arquivo deve existir e poder ser modificado;
  - **w+b** - abre (ou cria) um arquivo BINÁRIO para leitura e gravação, se o arquivo já existir, o conteúdo anterior será destruído;
  - **a+b** - abre (ou cria) um arquivo BINÁRIO para leitura e gravação, as escritas serão realizadas no fim do arquivo.

Exemplo da função *fopen*

```
FILE *fp;
char nomeArquivo[] = "arquivo.bin";
fp = fopen(nomeArquivo, "wb");
```

Buffers do SO

- Quando nós abrimos um arquivo (texto ou binário) com *fopen*, o SO **reserva** aquele arquivo para nosso programa, e aloca uma série de recursos para trabalharmos com ele.
- Visto que operações de acesso aos dispositivos de armazenamento (HD, fita, *pendrive*, etc...) são muito lentas, todo arquivo aberto possui associados a ele *buffers*, i.e., **áreas de memória temporárias**, de entrada e saída.
- Quando fazemos uma leitura, o SO traz o máximo possível de dados de uma vez só e armazena-os no ***buffer* de entrada**, onde ficam imediatamente à disposição para as leituras subsequentes.
- Quando escrevemos, os dados vão sendo acumulados no ***buffer* de saída**, antes que sejam efetivamente escritos no disco, o que só acontece quando:
  - o ***buffer* fica cheio**, ou seja, todo o espaço reservado é ocupado com dados;
  - o **arquivo é fechado** e os recursos são liberados;
  - o programador **solicita explicitamente**, por meio da função *fflush*;
- Por esse motivo, para garantir que os dados foram efetivamente guardados, é necessário utilizar *fflush* (se ainda formos usar o arquivo) ou *fclose* (se não vamos mais trabalhar com ele).

Arquivos Binários - Leitura e Escrita

- Em arquivos binários, lemos e escrevemos **blocos** de bytes, utilizando, respectivamente, as seguintes funções:

```
int fread (void *ptr, int tam, int cont, FILE *fp);

int fwrite (void *ptr, int tam, int cont, FILE *fp);
```

- Os parâmetros são os seguintes:
  - **ptr** – é um ponteiro para a região na memória de/para onde os dados serão lidos/escritos;
  - **tam** – é o tamanho (em *bytes*) de um bloco individual;
  - **cont** – é o número de blocos a serem lidos/escritos;
  - **fp** – é o ponteiro de arquivo.
- Observe que *tam x cont* indica o **total de *bytes*** a serem lidos/escritos. Para descobrir o tamanho de uma variável ou de um tipo, utilize a palavra chave **sizeof**, como veremos no próximo slide.
- Ambas as funções retornam o **total de blocos transferidos** (lidos/escritos) **com sucesso**. Podemos comparar esse valor com *cont* para descobrir se houve algum problema na leitura/escrita.

Arquivos Binários – Exemplo 1

Exemplo - Manipulando arquivos binários

```c
#include <stdio.h>

int main () {
    FILE *fp;
    char string_lida[50], string[50] = "Ola, mundo binario!!";
    char nomearq[50] = "arquivo.bin";

    fp = fopen(nomearq, "r+b");
    if (fp == NULL) {
        fp = fopen(nomearq, "wb");
    }

    fwrite(string, sizeof(string), 1, fp);
    fclose(fp);

    fp = fopen(nomearq, "rb");
    fread(string_lida, sizeof(string), 1, fp);
    printf("Veja o conteudo da string lida do arquivo: %s\n", string_lida);
    fclose(fp);

    getchar();
    return 0;
}
```

Arquivos Binários – Exemplo 1

Exemplo - Manipulando arquivos binários

```c
#include <stdio.h>

int main () {
    FILE *fp;
    char string[50] = "Ola, mundo binario!!";
    char string_lida[50];
    char nomearq[50] = "arquivo.bin";

    fp = fopen(nomearq, "r+b");
    if (fp == NULL) {
        fp = fopen(nomearq, "wb");
    }

    fwrite(string, sizeof(string), 1, fp);
    fclose(fp);

    fp = fopen(nomearq, "rb");
    fread(string_lida, sizeof(string), 1, fp);
    printf("Veja o conteudo da string lida do arquivo: %s\n", string_lida);
    fclose(fp);

    getchar();
    return 0;
}
```

O comando **sizeof(*<var ou tipo>*)** retorna o número de bytes ocupado por uma variável ou por um tipo.
Veja os exemplos:

```c
int tam_int = sizeof(int);
int tam_pessoa = sizeof(struct Pessoa);
int tam_vetor = sizeof(vetor);
```

Arquivos Binários – Exemplo 1

Exemplo - Manipulando arquivos binários

```c
#include <stdio.h>

int main () {
    FILE *fp;
    char string_lida[50], string[50] = "Ola, mundo binario!!";
    char nomearq[50] = "arquivo.bin";

    fp = fopen(nomearq, "r+b");
    if (fp == NULL) {
        fp = fopen(nomearq, "wb");
    }

    fwrite(string, sizeof(string), 1, fp);
    fclose(fp);

    fp = fopen(nomearq, "rb");
    fread(string_lida, sizeof(string), 1, fp);
    printf("Veja o conteudo da string lida do arquivo: %s\n", string_lida);
    fclose(fp);

    getchar();
    return 0;
}
```

Observe que podemos escrever e ler a *string* toda com um comando só, porque estamos guardando sua representação binária, como ela está na memória.

Exemplo - Gravando uma *struct*

```c
#include <stdio.h>

typedef struct {
    int cod;
    char nome[30];
} Pessoa;

int main () {
    FILE *fp;
    char nomearq[50] = "arquivo.bin";
    Pessoa pessoa;
    int i;

    fp = fopen(nomearq, "wb");
    for (i = 0; i < 3; i++) {
        printf("Informe o nome:\n");
        scanf("%s", pessoa.nome);
        printf("Informe o codigo:\n");
        scanf("%d", &pessoa.cod);
        fwrite(&pessoa, sizeof(Pessoa), 1, fp);
    }
    fclose(fp);

    fp = fopen(nomearq, "rb");
    while (fread(&pessoa, sizeof(Pessoa), 1, fp) != 0) {
        printf("Nome: %s\n", pessoa.nome);
        printf("Cod: %d\n", pessoa.cod);
        getchar();
    }
    fclose(fp);

    return 0;
}
```

Arquivos Binários – Exemplo 2

Exemplo - Gravando uma *struct*

```c
#include <stdio.h>

typedef struct {
    int cod;
    char nome[30];
} Pessoa;

int main () {
    FILE *fp;
    char nomearq[50] = "arquivo.bin";
    Pessoa pessoa;
    int i;

    fp = fopen(nomearq, "wb");
    for (i = 0; i < 3; i++) {
        printf("Informe o nome:\n");
        scanf("%s", pessoa.nome);
        printf("Informe o codigo:\n");
        scanf("%d", &pessoa.cod);
        fwrite(&pessoa, sizeof(Pessoa), 1, fp);
    }
    fclose(fp);

    fp = fopen(nomearq, "rb");
    while (fread(&pessoa, sizeof(Pessoa), 1, fp) != 0) {
        printf("Nome: %s\n", pessoa.nome);
        printf("Cod: %d\n", pessoa.cod);
        getchar();
    }
    fclose(fp);

    return 0;
}
```

Estamos solicitando do usuário os dados de 3 pessoas e gravando no arquivo.

Grava 1 pessoa por vez. Observe que passamos um ponteiro para a variável pessoa.

Abre o arquivo e lê os registros (Pessoa) um a um, mostrando os dados na tela. Quando a função *fread* retorna 0, significa que o arquivo chegou ao fim (ou houve algum erro de leitura).

Exemplo - Gravando um vetor de *structs*

```c
#include <stdio.h>

typedef struct {
    int cod;
    char nome[30];
} Pessoa;

int main () {
    FILE *fp;
    char nomearq[50] = "arquivo.bin";
    Pessoa funcionarios[3];
    int i;

    fp = fopen(nomearq, "wb");
    for (i = 0; i < 3; i++) {
        printf("Informe o nome:\n");
        scanf("%s", funcionarios[i].nome);
        printf("Informe o codigo:\n");
        scanf("%d", &funcionarios[i].cod);
    }
    fwrite(funcionarios, sizeof(Pessoa), 3, fp);
    fclose(fp);

    fp = fopen(nomearq, "rb");
    fread(funcionarios, sizeof(Pessoa), 3, fp)
    for (i = 0; i < 3; i++) {
        printf("Nome: %s\n", funcionarios[i].nome);
        printf("Cod: %d\n", funcionarios[i].cod);
        getchar();
    }
    fclose(fp);

    return 0;
}
```

Arquivos Binários – Exemplo 3

Exemplo - Gravando um vetor de *structs*

```c
#include <stdio.h>

typedef struct {
    int cod;
    char nome[30];
} Pessoa;

int main () {
    FILE *fp;
    char nomearq[50] = "arquivo.bin";
    Pessoa funcionarios[3];
    int i;

    fp = fopen(nomearq, "wb");
    for (i = 0; i < 3; i++) {
        printf("Informe o nome:\n");
        scanf("%s", funcionarios[i].nome);
        printf("Informe o codigo:\n");
        scanf("%d", &funcionarios[i].cod);
    }
    fwrite(funcionarios, sizeof(Pessoa), 3, fp);
    fclose(fp);

    fp = fopen(nomearq, "rb");
    fread(funcionarios, sizeof(Pessoa), 3, fp)
    for (i = 0; i < 3; i++) {
        printf("Nome: %s\n", funcionarios[i].nome);
        printf("Cod: %d\n", funcionarios[i].cod);
        getchar();
    }
    fclose(fp);

    return 0;
}
```

Solicita e armazena os dados de 3 pessoas em um vetor de registros.

Grava o vetor inteiro (os 3 registros) de uma vez.

Lê o vetor inteiro (os 3 registros) de uma vez.

Posição Corrente no Arquivo

- Para todo arquivo, temos a noção de **posição corrente**.
- Esse valor inteiro indica o **próximo ponto** (onde zero é o começo do arquivo) em que haverá uma leitura ou escrita de dados.
- Quando um arquivo é aberto, a posição corrente é colocada no **início** do arquivo, se abrirmos com “r” ou “w”; ou no **final**, se abrirmos com “a”.
- A cada operação de leitura ou escrita (*fread*, *fwrite*, *fprintf*, *fscanf*), a posição corrente é **automaticamente atualizada**, avançando *N* passos, onde *N* é o total de *bytes* lidos/escritos.

Alterando a Posição Corrente

- Em geral, se vamos ler ou escrever **sequencialmente** no arquivo, não precisamos nos preocupar com a posição corrente, já que ela muda automaticamente.
- No entanto, frequentemente precisamos acessar blocos no arquivo **aleatoriamente**. Nesse caso, é necessário **alterar a posição corrente** antes de ler ou escrever. Para isso, fazemos:

```
int fseek (FILE *fp, int deslocamento, int cod_origem);
```

- Essa função recebe os seguintes parâmetros:
  - **`fp`** - o ponteiro de arquivo;
  - **`deslocamento`** - a quantidade de *bytes* (que pode ser negativa) a ser deslocada em relação à origem;
  - **`cod_origem`** - um código que indica o ponto de referência a partir do qual definir a posição (origem).
- O *cod_origem* pode receber um dos 3 valores abaixo:
  - **`SEEK_SET`** - para realizar o deslocamento a partir do início do arquivo;
  - **`SEEK_CUR`** - para realizar o deslocamento a partir da posição atual no arquivo;
  - **`SEEK_END`** - para realizar o deslocamento a partir do fim do arquivo;
- Veja o próximo exemplo...

Exemplo - Acessando um registro específico

```c
nclude <stdio.h>

pedef struct {
  int cod;
  char nome[30];
Pessoa;

t main () {
  FILE *fp;
  char nomearq[50] = "arquivo.bin";
  Pessoa funcionarios[5], pessoa;
  int i;

  fp = fopen(nomearq, "wb");
  for (i = 0; i < 5; i++) {
      printf("Informe o nome:\n");
      scanf("%s", funcionarios[i].nome);
      printf("Informe o codigo:\n");
      scanf("%d", &funcionarios[i].cod);
  }
  fwrite(funcionarios, sizeof(Pessoa), 5, fp);
  fclose(fp);

  fp = fopen(nomearq, "rb");
  fseek(fp, 2 * sizeof(Pessoa), SEEK_SET);
  fread(&pessoa, sizeof(Pessoa), 1, fp);
  printf("Nome: %s\n", pessoa.nome);
  printf("Cod: %d\n", pessoa.cod);
  fclose(fp);

  getchar();
  return 0;
```

Arquivos Binários – Exemplo 4

Exemplo - Acessando um registro específico

```c
#include <stdio.h>

typedef struct {
    int cod;
    char nome[30];
} Pessoa;

int main () {
    FILE *fp;
    char nomearq[50] = "arquivo.bin";
    Pessoa funcionarios[5], pessoa;
    int i;

    fp = fopen(nomearq, "wb");
    for (i = 0; i < 5; i++) {
        printf("Informe o nome:\n");
        scanf("%s", funcionarios[i].nome);
        printf("Informe o codigo:\n");
        scanf("%d", &funcionarios[i].cod);
    }
    fwrite(funcionarios, sizeof(Pessoa), 5, fp);
    fclose(fp);

    fp = fopen(nomearq, "rb");
    fseek(fp, 2 * sizeof(Pessoa), SEEK_SET);
    fread(&pessoa, sizeof(Pessoa), 1, fp);
    printf("Nome: %s\n", pessoa.nome);
    printf("Cod: %d\n", pessoa.cod);
    fclose(fp);

    getchar();
    return 0;
}
```

Solicita e armazena os dados de 5 pessoas em um vetor de registros.

Grava o vetor inteiro (5 registros) de uma vez.

Posiciona o arquivo no começo do 3º registro (contando a partir do início do arquivo). Note que, da mesma forma que com vetores, começamos a contar do zero, logo, o 3º registro começa no *byte* 2 * (tamanho de um registro).

Lê 1 registro na posição atual, ou seja, o 3º registro.

Verificando o Estado do Arquivo

- É uma boa prática sempre **verificar o estado** do arquivo após cada leitura/escrita.
- Quando fazemos uma chamada a uma das funções *fread*, *fwrite*, *fprintf* ou *fscanf* e ocorre algum erro ou o arquivo chega ao fim, esta informação **fica registrada** no ponteiro (fp), e podemos verificá-la utilizando:

```
feof(FILE *fp);
ferror(FILE *fp);
```

- A primeira função retorna um valor diferente de zero **se o arquivo está no fim**. A segunda, retorna diferente de zero **se ocorreu algum erro**.
- Vamos ver um exemplo.

Exemplo - Gravando um vetor de *structs*

```c
#include <stdio.h>

typedef struct {
    int cod;
    char nome[30];
} Pessoa;

int main () {
    FILE *fp;
    char nomearq[50] = "arquivo.bin";
    Pessoa pessoa;
    int cont, erro;
    /* tenta abrir o arquivo para leitura */
    fp = fopen(nomearq, "rb");
    if (fp == NULL) {
        printf("Erro ao abrir o arquivo de entrada.\n");
        return -1;
    }
    /* le registros enquanto nao chegar ao final do arquivo ou ocorrer algum erro */
    cont = 0;
    while (fread(&pessoa, sizeof(Pessoa), 1, fp) != 0) {
        cont++;
        printf("Nome: %s\n", pessoa.nome);
        printf("Cod: %d\n\n", pessoa.cod);
    }
    erro = 0;
    if (feof(fp))
        printf("Foram lidos %d registros ao todo.\n", cont);
    else if (ferror(fp)) {
        printf("Ocorreu um erro ao ler o arquivo!\n");
        erro = -1;
    }
    fclose(fp);
    getchar();
    return erro;
}
```

O programa sai do loop (*i.e.*, *fread* retorna zero) quando não há mais registros a serem lidos ou se houver algum erro. Aqui, estamos verificando essas duas condições e, se for o caso, setamos erro = -1 para retornar um código de erro na função main.