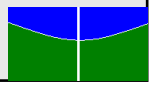
COMPUTAÇÃO BÁSICA

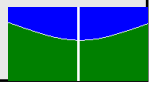
Disciplina: 116301

Profa. Carla Denise Castanho

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



15. ARQUIVOS



Arquivos

- Os comandos de entrada e saída que usamos até o momento foram:
  - printf – mostra dados formatados na saída padrão
  - scanf – lê dados formatados da entrada padrão
- Todavia, dados podem ser lidos e gravados em arquivos (em geral, em discos).
- Um arquivo em disco é um espaço com diversas características físicas que dependem do tipo de dispositivo que é usado (disquete, disco rígido, CD, etc.). Porém, utilizando uma abstração, um arquivo em disco pode ser visto como um espaço sequencial (*stream*) onde são lidos ou gravados os dados.
- Cada arquivo está associado a um nome, pelo qual o mesmo é conhecido externamente, isto é, o nome que consta no sistema operacional.
- Uma vez que um arquivo é uma sequência de bytes, um valor especial chamado de **EOF (EndOfFile)** é utilizado para marcar o final de um arquivo.

Arquivos

- A maioria dos sistemas operacionais divide os arquivos em dois tipos:

- TEXTO:
  - são gravados caracteres, ou seja, letras formando um texto de fato. Um código fonte em linguagem C, por exemplo, é um arquivo tipo texto, que podemos visualizar facilmente com um editor de textos.

- BINÁRIO:
  - são gravados os dados como estariam na memória, byte a byte. Por exemplo, uma variável inteira é gravada com 4 bytes com o conteúdo exato que está na memória. Não conseguimos visualizar com um editor de texto, é necessário um programa que reconheça aquela sequência de bytes e dê significado a ela.

Trabalhando com Arquivos

- Para trabalhar com arquivos, a linguagem C fornece uma camada de abstração entre o programador e o dispositivo físico onde está gravado o arquivo.
- Esta abstração é chamada fila de bytes e o dispositivo normalmente é o disco, mas pode ser inclusive um conceito abstrato como a *saída padrão*. Existe um sistema *bufferizado* de acesso ao arquivo, onde um **ponteiro de arquivo** define vários aspectos do arquivo, como nome, status e posição corrente, além de ter a fila associada a ele.

  Assim criamos um:

```
FILE *fp;
```

  Ou seja, um ponteiro *fp*, do tipo arquivo. Note-se que, como qualquer outra variável em C, o nome *fp* pode ser alterado.

Trabalhando com Arquivos

- Antes de ler ou gravar em algum arquivo precisamos ter certeza de algumas coisas:
  - Onde se encontra o arquivo?
  - O arquivo já existe, e vamos apenas abrí-lo?
  - O arquivo não existe e vamos criá-lo?
  - O arquivo já existe mas vamos recriá-lo?

- Assim, temos a função ***fopen*** (*file open*), que possui dois parâmetros, nessa ordem: o nome do arquivo e o tipo de operação que faremos com ele.
- Por exemplo:

```
FILE *fp;
char nomeArquivo[] = "c:\\arquivo.txt";
fp = fopen(nomeArquivo, "w");
```


Trabalhando com Arquivos

- Caso a função ***fopen*** não encontre o arquivo indicado, ela retorna *NULL*. Dessa forma, podemos testar se o arquivo já existe antes de criar um novo, o que recriaria (apagaria os dados) do arquivo existente:

```
FILE *fp;
char nomeArquivo[] = "c:\\arquivo.txt";
fp = fopen(nomeArquivo, "r+");
/* caso o arquivo não exista, cria um novo */
if (fp == NULL) {
    fp = fopen(nomeArquivo, "w");
}
```


Trabalhando com Arquivos

- Tabela de funcionalidades:
  - r - abre um arquivo TEXTO para leitura
  - w - cria um arquivo TEXTO para gravação, ou, se o arquivo já existe, elimina seu conteúdo e recomeça a gravação a partir do seu início.
  - a - abre um arquivo TEXTO já existente para gravação, e sempre grava a partir de seu final.
  - rb - abre um arquivo BINÁRIO para leitura
  - wb - cria um arquivo BINÁRIO para gravação, ou, se o arquivo já existe, elimina seu conteúdo e recomeça a gravação a partir do seu início.
  - ab - abre um arquivo BINÁRIO já existente para gravação, e sempre grava a partir de seu final.
  - r+ - Abre um arquivo TEXTO para leitura e gravação. O arquivo deve existir e pode ser modificado.
  - w+ - Cria um arquivo TEXTO para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado.

Trabalhando com Arquivos

- Tabela de funcionalidades:
  - a+ - Abre um arquivo TEXTO para gravação e leitura. Os dados serão adicionados no fim do arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não exista.
  - r+b - Abre um arquivo BINÁRIO para leitura e escrita. O mesmo que "r+" acima, só que o arquivo é binário.
  - w + b - Cria um arquivo BINÁRIO para leitura e escrita. O mesmo que "w+" acima, só que o arquivo é binário.
  - a + b - Acrescenta dados ou cria uma arquivo BINÁRIO para leitura e escrita. O mesmo que "a+" acima, só que o arquivo é binário.

Trabalhando com Arquivos

- Da mesma forma que devemos abrir um arquivo utilizando a função ***fopen***, devemos fechá-lo quando não formos mais utilizá-lo, liberando recursos do SO.
- Ao fecharmos o arquivo, também garantimos que o mesmo será salvo em disco, e não ficará simplesmente no buffer (região de memória).
- Para isso, utilizamos o comando ***fclose***, da seguinte forma:

```
fclose(fp);  /* fp é o ponteiro para o arquivo*/
```

- Para simplesmente salvar os dados em disco, sem fechar o arquivo, utilizamos o comando ***fflush***, da seguinte forma:

```
fflush(fp);  /* fp é o ponteiro para o arquivo*/
```


Trabalhando com Arquivos

- Muitas vezes precisamos reposicionar o ponteiro no inicio do arquivo. Para isso, podemos utilizar a função ***rewind***, da seguinte forma:

```
rewind(fp); /*retorna a posição corrente do arquivo
  para o início*/
```


Arquivo Texto

- Para fazermos a gravação e leitura em um arquivo **texto**, devemos utilizar as funções ***fprintf***, e ***fscanf***, respectivamente.
- Essas funções têm como parâmetros, nessa ordem: o ponteiro do arquivo; uma *string* descrevendo os tipos dos dados; e as variáveis (ou endereços das variáveis) que utilizaremos para representar esses valores.
  - Por exemplo, para escrever números inteiros em um arquivo:

```
int numero1, numero2, numero3;
fprintf(fp, "%d %d %d", numero1, numero2, numero3);
```

  - Para ler números inteiros de um arquivo:

```
fscanf(fp, "%d %d %d", &numero1, &numero2,
&numero3);
```


Arquivo Texto

Exemplo: um programa que escreve a string “Meu primeiro programa com arquivo texto” em um arquivo; em seguida, lê a mesma (do arquivo) e mostra seu conteúdo na tela:

```
#include <stdio.h>
#include <string.h>

int main (){
        FILE *fp;
        char string[50] = “Meu primeiro programa com arquivo  texto”;
        char nomeArquivo[50] = “arquivo.txt”;
            fp = fopen(nomeArquivo, "r+“);
        if (fp == NULL) {
                fp = fopen(nomeArquivo, "w");
        }
        fprintf(fp, "%s", string);
        fclose(fp);
        fopen(nomeArquivo,"r+");
        fscanf(fp, "%s", string);
        printf("string do arquivo e: %s\n", string);
        getchar();
        fclose(fp);
        return 0;
}
```

Tenta abrir o arquivo para gravação

Testa se o arquivo existe

Caso não exista, cria e abre p/ gravação

Grava no arquivo

Fecha o arquivo

Lê do arquivo

Fecha o arquivo

Abre o arquivo

Arquivo Texto

Exemplo: um programa que escreve a string “Meu primeiro programa com arquivo texto” em um arquivo; em seguida, lê a mesma (do arquivo) e mostra seu conteúdo na tela:

```
#include <stdio.h>
#include <string.h>

int main (){
        FILE *fp;
        char string[50] = “Meu primeiro p
        char nomeArquivo[50] = “arquivo
            fp = fopen(nomeArquivo, "r+“
        if (fp == NULL) {
                fp = fopen(nomeArquivo, "w");
        }
        fprintf(fp, "%s", string);
        fclose(fp);
        fopen(nomeArquivo,"r+");
        fscanf(fp, "%s", string);
        printf("string do arquivo e: %s\n", string);
        getchar();
        fclose(fp);
        return 0;
}
```

Uma vez que fechamos o arquivo e o reabrimos, não há necessidade de usarmos o rewind() para o ponteiro voltar à posição original, pois quando abrimos o arquivo ele já está lá.

Arquivo Texto

- OBS sobre o exemplo do slide anterior:
  - Note que a saída resultante é simplesmente a palavra *Meu*, pois, em um arquivo do tipo texto, os espaços indicam um novo registro (dado).
  - Para solucionar isso, temos duas alternativas:
    1. Fazer um loop para ler as strings do arquivo enquanto não chegar no final do mesmo (veja exemplo no próximo slide)
    2. Utilizar o comando fscanf da seguinte forma:

       ```
       fscanf(fp, "%[^\n]s", string);
       ```

       (veja o exemplo dois slides adiante)
  - Observe que podemos abrir o *arquivo.txt* usando um editor de textos e é possível visualizar o conteúdo que foi escrito nele.

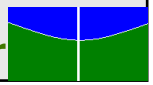
Arquivo Texto

Exemplo: um programa que escreve a string “Meu primeiro programa com arquivo texto” em um arquivo; em seguida, lê a mesma (do arquivo) e mostra seu conteúdo na tela:

```
#include <stdio.h>
#include <string.h>

int main (){
        FILE *fp;
        char string[50] ="Meu primeiro programa com arquivo texto";
        char nomeArquivo[50] = "arquivo.txt";
        fp = fopen(nomeArquivo, "r+");
        if (fp == NULL) {
              fp = fopen(nomeArquivo, "w");
        }
        fprintf(fp, "%s", string);
        fclose(fp);
        fopen(nomeArquivo,"r+");
        while (fscanf(fp, "%s", string) > 0) {
              printf("%s ", string); getchar();
        }
        fclose(fp);
        return 0;
}
```

Note que cada vez que chamamos a funcao *fscanf* ela incrementa automaticamente o ponteiro do arquivo



Arquivo Texto

Exemplo: um programa que escreve a string “Meu primeiro programa com arquivo texto” em um arquivo; em seguida, lê a mesma (do arquivo) e mostra seu conteúdo na tela:

```
#include <stdio.h>
#include <string.h>

int main (){
        FILE *fp;
        char string[50] ="Meu primeiro programa com arquivo texto";
        char nomeArquivo[50] = "arquivo.txt";
        fp = fopen(nomeArquivo, "r+");
        if (fp == NULL) {
              fp = fopen(nomeArquivo, "w");
        }
        fprintf(fp, "%s", string);
        fclose(fp);
        fopen(nomeArquivo,"r+");
        fscanf(fp, "%[^\n]s", string);
        printf(string);
        fclose(fp);
        return 0;
}
```

Lê e escreve toda a string de uma vez.

Arquivos Binários

- **ARQUIVOS BINÁRIOS**: São arquivos versáteis, com maior flexibilidade de leitura e escrita, pois podemos gravar estruturas inteiras de uma vez, como, por exemplo, vetores, e acessá-los facilmente; ao contrário dos arquivos TEXTO em que temos de ler (e escrever) dado por dado.
- Com arquivos binários, usamos basicamente as funções ***fwrite()*** para escrever e ***fread()*** para ler.

Trabalhando com Arquivos

- **ARQUIVOS BINÁRIOS**

```
fwrite(&nome_da_variavel, sizeof(tipo_variavel),
   nro_de_elementos, ponteiro_de_arquivo)
/*o comando sizeof retorna o tamanho, em bytes, do tipo da
   variavel*/


fread(&nome_da_variavel, sizeof(tipo_variavel),
   nro_de_elementos, ponteiro_de_arquivo)
```

- No caso da função ***fwrite***, a variável é aquela onde o dado a ser **gravado** se encontra e que será passado para o arquivo.
- No caso da função ***fread*** , é a variável que irá receber o valor que será **lido** do arquivo.

Arquivo Binário – Exemplo 1

Exemplo: um programa que escreve a string “Meu primeiro programa com arquivo binário” em um arquivo; em seguida, lê a mesma (do arquivo) e mostra seu conteúdo na tela:

```
#include <stdio.h>
#include <string.h>

int main (){
        FILE *fp;
        char string[50] = "Meu primeiro programa com arquivo binario";
        char nomeArquivo[50] = "arquivoBinario.bin";
          fp = fopen(nomeArquivo, "a+b");
        if (fp == NULL) {
              fp = fopen(nomeArquivo, "wb");
        }
        fwrite(string, sizeof(string),1,fp);
        fclose(fp);
        fp = fopen(nomeArquivo,"a+b");
        fread(string, sizeof(string),1,fp);
        printf("Veja o conteudo da string lida do arquivo:\n");
        printf(string);
           getchar();
        fclose(fp);
        return 0;
}
```

Note que foi possível escrever toda a string sem o WHILE. Isso foi possivel pois foi guardada no arquivo a estrutura binária da string.

Arquivo Binário – Exemplo 2

Exemplo: um programa que **escreve uma STRUCT em um arquivo**, e depois lê esta informação do arquivo e mostra na tela. Neste exemplo são lidas e gravadas informações (código e nome) para 3 pessoas.

```
#include <stdio.h>

typedef  struct {
        int codigo;
        char nome[30];
} tipoDadosPessoa;

int main() {
      FILE *fp;
      tipoDadosPessoa dadosDePessoa;
      int i;
      char nomeArquivo[50] = "arqbin.bin";
      fp = fopen(nomeArquivo, "a+b");
      if (fp == NULL) {
            fp = fopen(nomeArquivo, "wb");
      }

/* CONTINUA NO PRÓXIMO SLIDE */
```


Arquivo Binário – Exemplo 2

/* CONTINUAÇÃO DO SLIDE ANTERIOR */

```
        for (i = 0; i<3; i++ ) {
                printf("Informe o nome: ");
                scanf("%s", dadosDePessoa.nome);
                printf("Informe o cod: ");
                scanf("%d", &dadosDePessoa.codigo);
                fwrite(&dadosDePessoa,sizeof(tipoDadosPessoa),1
        }
        fclose(fp);
        fopen(nomeArquivo,"a+b");
        while (fread(&dadosDePessoa, sizeof(tipoDadosPessoa),1,fp) != 0) {
                printf("\n\nNome: %s",dadosDePessoa.nome);
                printf("\nCod: %d",dadosDePessoa.codigo);
                getchar();
        }
        fclose(fp);
}
```

Lê um registro e grava no arquivo.

Lê um registro de cada vez do arquivo.
O loop encerra quando a função ***fread*** retornar 0 (zero) elementos lidos, ou seja, quando chegar no fim do arquivo.

Arquivo Binário – Exemplo 3

Exemplo: um programa que escreve **um vetor de STRUCT (registros)** um arquivo, e depois lê esta informação do arquivo e mostra na tela. Neste exemplo são lidas e gravadas informações (código e nome) para 3 pessoas.

```
#include <stdio.h>

typedef  struct {
            int codigo;
            char nome[30];
} tipoDadosDeFuncionario;

int main() {
      FILE *fp;
      tipoDadosDeFuncionario dadosDeFuncionario[3], dadoslidos[3];
      int i;
      char nomeArquivo[50] = "arqbin.bin";

      fp = fopen(nomeArquivo, "a+b");
      if (fp == NULL) {
            fp = fopen(nomeArquivo, "wb");
      }

/* CONTINUA NO PRÓXIMO SLIDE */
```


Arquivo Binário – Exemplo 3

/* CONTINUAÇÃO DO SLIDE ANTERIOR*/

```
      for (i = 0; i<3; i++ ) {
            printf("Informe o nome: ");
            scanf("%s", dadosDeFuncionario[i].nome);
            printf("Informe o cod: ");
            scanf("%d", &dadosDeFuncionario[i].codigo);
       }
      fwrite(dadosDeFuncionario,sizeof(tipoDadosDeFuncionario),3,fp);
      fclose(fp);
      fp = fopen(nomeArquivo,"a+b");
      fread(&dadosLidos, sizeof(tipoDadosDeFuncionario),3,fp);
      for (i = 0; i<3; i++ ) {
          printf("\n\nNome: %s",dadosLidos[i].nome);
          printf("\nCod: %d",dadosLidos[i].codigo);
          getchar();
          getchar();
      }
}
```

Grava no arquivo de uma vez só todo o vetor com 3 registros.

Lê 3 registros de uma vez do arquivo, e grava direto os 3 no vetor dadoslidos.

Arquivo Binário

- Note que temos a facilidade na manipulação de arquivos binários, todavia tente abrir o arquivoBinario.bin no notepad e veja que não conseguimos lê-lo facilmente!

Arquivos

- ATENÇÃO:
- Toda vez que estamos trabalhando com arquivos, há uma espécie de posição atual no arquivo. Esta é a posição de onde será lido ou escrito o próximo caractere. Normalmente, num acesso sequencial a um arquivo, não temos que mexer nesta posição pois quando lemos um caractere a posição no arquivo é automaticamente atualizada. Num acesso randômico teremos que mexer nesta posição (ver função fseek() a seguir).

Trabalhando com Arquivos

- ***fseek():*** para se fazer procuras e acessos randômicos em arquivos, usa-se a função ***fseek()***. Ela move a posição corrente de leitura ou escrita no arquivo para um deslocamento especificado, a partir de um ponto dado.

  fseek (FILE *fp, deslocamento em bytes,origem);

  - O parâmetro *origem* determina a partir de onde os *deslocamento em bytes* de movimentação serão contados. Os valores possíveis são definidos por macros em **stdio.h** e são:
    - SEEK_SET = 0 = Início do arquivo
    - SEEK_CUR = 1 = Ponto corrente no arquivo
    - SEEK_END = 2 = Fim do arquivo
  - Tendo-se definido a partir de onde irá se contar, *deslocamento em bytes* determina quantos bytes de deslocamento serão dados na posição atual.

Arquivo Binário – Exemplo 4 – fseek()

Exemplo: um programa que escreve **um vetor de STRUCT (registros)** um arquivo com informações (código e nome) para 5 pessoas. Depois o programa acessa, lê, e mostra o terceiro registro gravado no arquivo.

```
#include <stdio.h>

typedef  struct {
            int codigo;
            char nome[30];
} tipoDadosDeFuncionario;

int main() {
      FILE *fp;
      tipoDadosDeFuncionario dadosDeFuncionario[5], dadoslidos;
      int i;
      char nomeArquivo[50] = "arqbin.bin";

      fp = fopen(nomeArquivo, "a+b");
      if (fp == NULL) {
            fp = fopen(nomeArquivo, "wb");
      }

/* CONTINUA NO PRÓXIMO SLIDE */
```


Arquivo Binário – Exemplo 4 – fseek()

/* CONTINUAÇÃO DO SLIDE ANTERIOR*/

```
for (i = 0; i<5; i++ ) {
        printf("Informe o nome: ");
        scanf("%s", dadosDeFuncionario[i].nome);
        printf("Informe o cod: ");
        scanf("%d", &dadosDeFuncionario[i].codigo);
}
fwrite(&dadosDeFuncionario,sizeof(tipoDadosDeFuncionario),5,fp);
fclose(fp);
fopen(nomeArquivo,"a+b");
fseek (fp,(sizeof(tipoDadosDeFuncionario)*2),0);
fread(&dadoslidos, sizeof(tipoDadosDeFuncionario),1,fp);
printf("\n\nNome: %s",dadoslidos.nome);
printf("\nCod: %d",dadoslidos.codigo);
getchar();
getchar();
}
```

Grava no arquivo de uma vez só todo o vetor com 5 registros.

Posiciona o ponteiro do arquivo no início do 3o registro.

Lê um registro a partir da posição onde o ponteiro se encontra, no caso o 3o registro.

Trabalhando com Arquivos

- Um bom site para consultas rápidas é:

  http://www.mtm.ufsc.br/~azeredo/cursoC/c.html