Programação Estruturada

Arquivos

Professores Emílio Francesquini e Carla Negri Lintzmayer

2018.Q3

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

Parâmetros do programa: argc e argv

Argc e argv

- Até então temos criado programas onde a função `main()` não tem parâmetros.
- Mas esta função pode receber dois parâmetros: `main(int argc, char *argv[])`.
  - `argc` (*argument counter*): indica o número de argumentos na linha de comando ao se executar o programa.
  - `*argv[]` (*argument vector*): é um vetor de ponteiros para caracteres (ou seja vetor de strings) que contém os argumentos da linha de comando, um em cada posição do vetor.

Argc e argv

O programa abaixo imprime cada um dos parâmetros passados na linha de comando:

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    int i;
    for (i = 0; i < argc; i++) {
        printf("%s\n", argv[i]);
    }
    return 0;
}
```

Argc e argv

- Seu uso é útil em programas onde dados de entrada são passados via linha de comando.
- Exemplo: dados a serem processados estão em um arquivo, cujo nome é passado na linha de comando.

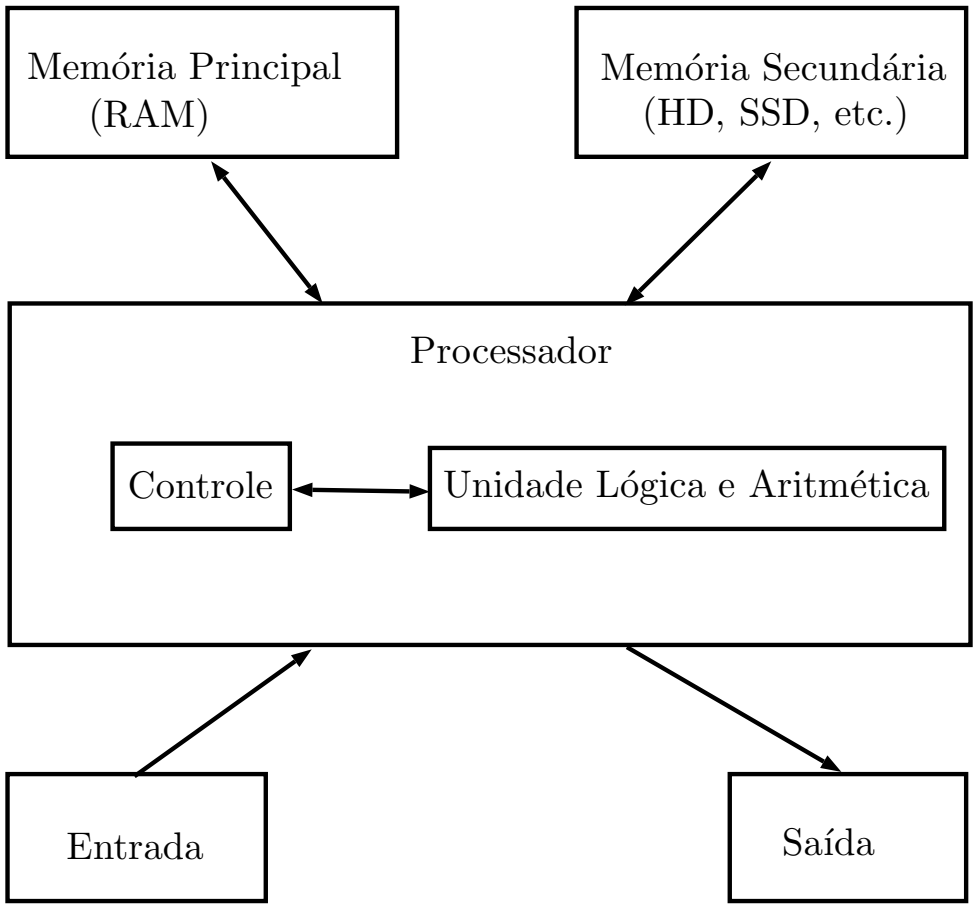
```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc < 2)
        printf("Informe o nome do arquivo.\n");
    else
        printf("Arquivo a ser processado: %s\n", argv[1]);
    return 0;
}
```

Arquivos

Tipos de memória

- Quando vimos a organização básica de um sistema computacional, havia somente um tipo de memória.
- Mas na maioria dos sistemas, a memória é dividida em dois tipos:



Tipos de memória

A memória principal (Random Access Memory) utilizada na maioria dos computadores, usa uma tecnologia que requer alimentação constante de energia para que informações sejam preservadas.

Tipos de memória

A memória secundária (como Hard Disks ou SSD) utilizada na maioria dos computadores, usa uma outra tecnologia que NÃO requer alimentação constante de energia para que informações sejam preservadas.

Tipos de memória

- Todos os programas executam na RAM, e por isso quando o programa termina ou acaba energia, as informações do programa são perdidas.
- Para podermos gravar informações de forma *persistente*, devemos escrever estas informações em arquivos na memória secundária.
- A memória secundária possui algumas características:
  - É muito mais lenta que a RAM.
  - É muito mais barata que a memória RAM.
  - Possui maior capacidade de armazenamento.
- Sempre que nos referirmos a um arquivo, estamos falando de informações armazenadas em memória secundária.

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)
arq.exe	arquivo executável (Windows)

Tipos de arquivos

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

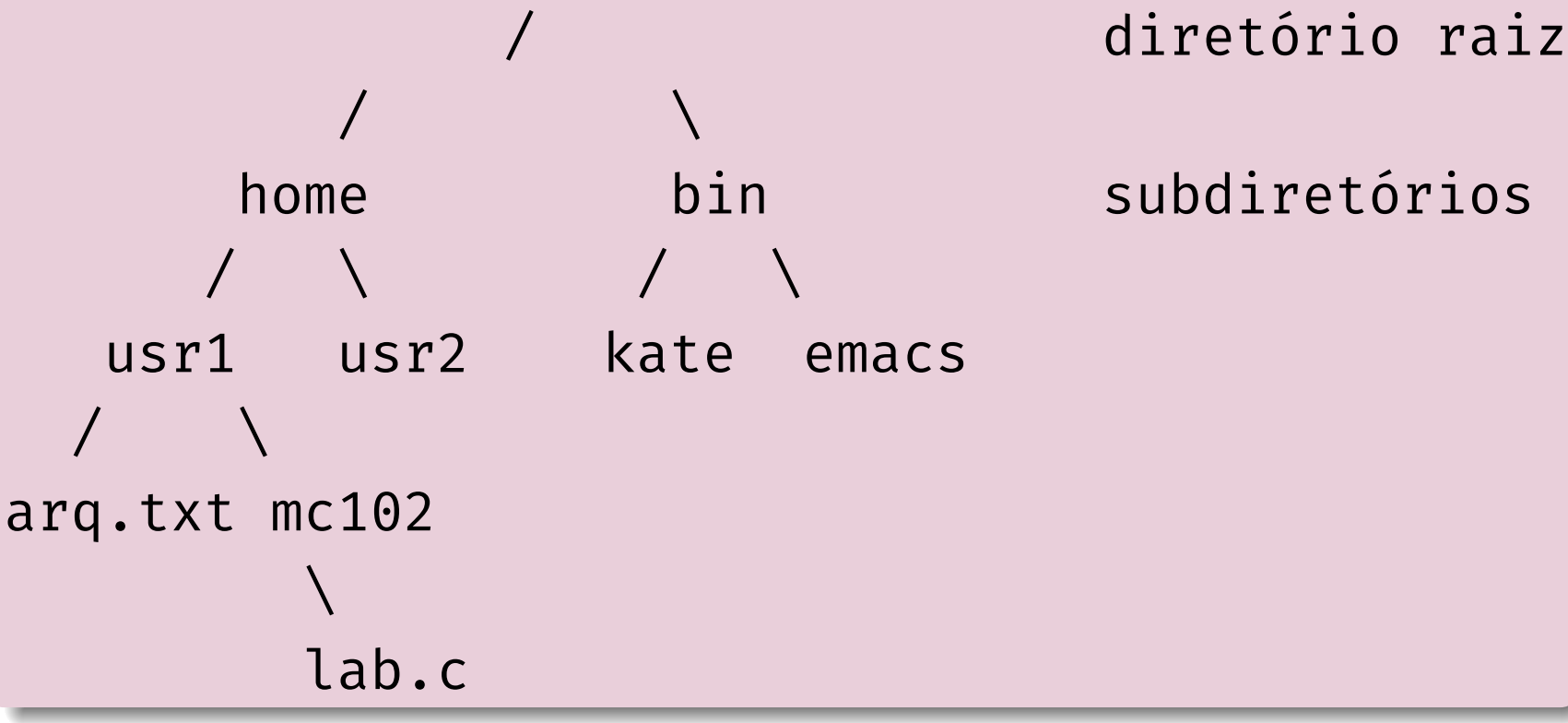
Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código fonte C, documento texto simples, páginas HTML.

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 e relativos

- O nome de um arquivo pode conter o seu diretório, ou seja, o caminho para encontrar este arquivo a partir da raiz.
- Desta forma o acesso a um arquivo pode ser especificado 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 a partir do diretório corrente.

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

Arquivos textos

Arquivos texto em C

- Em C, para se trabalhar com arquivos devemos criar um ponteiro especial: **um ponteiro para arquivos.**

```
FILE *nome_variavel;
```

- O comando acima cria um ponteiro para arquivos, cujo nome da variável é o nome especificado.

- Após ser criado um ponteiro para arquivo, podemos associá-lo com um arquivo real do computador usando a função **fopen**.

```
FILE *arq1;
arq1 = fopen("teste.txt", "r");
```

Arquivos texto em C

```
1 FILE *arq1;
2 arq1 = fopen("teste.txt", "r");
```

- O primeiro parâmetro para **fopen** é uma string com o nome do arquivo
  - Pode ser absoluto, por exemplo: ”/user/eduardo/teste.txt”
  - Pode ser relativo como no exemplo acima: ”teste.txt”
- O segundo parâmetro é uma string informando como o arquivo será aberto.
  - Se para leitura ou gravação de dados, ou ambos.
  - Se é texto ou se é binário.
  - No nosso exemplo, o **“r”** significa que abrimos um arquivo texto para leitura.

Abrindo um arquivo texto para leitura

- Antes de acessar um arquivo, devemos abri-lo com a função `fopen()`.
- A função retorna um ponteiro para o arquivo em caso de sucesso, e em caso de erro a função retorna `NULL`.
- Abrindo o arquivo `teste.txt`:

```
1  FILE *arq = fopen("teste.txt", "r");
2  if (arq == NULL)
3      printf("Erro ao tentar abrir o arquivo
       ↪  teste.txt.\n");
4  else
5      printf("Arquivo aberto para leitura.\n");
```

Lendo dados de um arquivo texto

- Para ler dados do arquivo aberto, usamos a função **fscanf()**, que é semelhante à função **scanf()**.
  - **int fscanf(ponteiro para arquivo, string de formato, variáveis)**.
  - A única diferença para o **scanf**, é que devemos passar como primeiro parâmetro um ponteiro para o arquivo de onde será feita a leitura.
- Lendo dados do arquivo `teste.txt`:

```
1 char aux;
2 FILE *f = fopen("teste.txt", "r");
3 fscanf(f, "%c", &aux);
4 printf("%c", aux);
```

Lendo dados de um arquivo texto

- Quando um arquivo é aberto, um **indicador de posição** no arquivo é criado, e este recebe a posição do início do arquivo.
- Para cada dado lido do arquivo, este indicador de posição é automaticamente incrementado para o próximo dado não lido.
- Eventualmente o indicador de posição pode chegar ao fim do arquivo:
  - A função **fscanf** devolve um valor especial, **EOF** (*End Of File*), caso tente-se ler dados e o indicador de posição está no fim do arquivo.

Lendo dados de um arquivo texto

- Para ler todos os dados de um arquivo texto, basta usarmos um laço que será executado enquanto não chegarmos no fim do arquivo.
- Lendo dados do arquivo `teste.txt`:

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

- O comando **fclose** (no fim do código) deve sempre ser usado para fechar um arquivo que foi aberto.
  - Quando escrevemos dados em um arquivo, este comando garante que os dados serão efetivamente escritos no arquivo.

Lendo dados de um arquivo texto: exemplo

Programa que imprime o conteúdo de um arquivo passado como parâmetro do programa:

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    FILE *arq;
    char aux;
    if (argc <2) {
        printf("Informe o nome do arquivo.\n");
        return 1;
    }
    arq = fopen(argv[1], "r");
    if (arq == NULL) {
        printf("Problema ao ler o arquivo.\n");
        return 1;
    }
    while (fscanf(arq, "%c", &aux) != EOF) {
        printf("%c", aux);
    }
    fclose(arq);
```

Lendo dados de um arquivo texto

- Notem que ao realizar a leitura de um caractere, automaticamente o indicador de posição do arquivo se move para o próximo caractere.
- Ao chegar no fim do arquivo a função **fscanf** retorna o valor especial **EOF**.
- Para voltar ao início do arquivo você pode fechá-lo e abrí-lo mais uma vez, ou usar o comando **rewind**.

```
1  while (fscanf(arq, "%c", &aux) != EOF)
2      printf("%c", aux);
3
4  printf("\n\n -----Imprimindo novamente\n\n");
5  rewind(arq);
6
7  while (fscanf(arq, "%c", &aux) != EOF)
8      printf("%c", aux);
```

Escrevendo dados em um arquivo texto

- Para escrever em um arquivo, ele deve ser aberto de forma apropriada, usando a opção **w**.
- Usamos a função **fprintf()**, semelhante à função **printf()**.
  - **int fprintf(ponteiro para arquivo, string de saída, variáveis)**
  - É semelhante ao **printf** mas notem que precisamos passar o ponteiro para o arquivo onde os dados serão escritos.
- Copiando dois arquivos:

```
1  FILE *arqin = fopen("entrada.txt", "r");
2  FILE *arqout = fopen("saida.txt", "w");
3  char aux;
4  while (fscanf(arqin, "%c", &aux) != EOF)
5      fprintf(arqout, "%c", aux);
6  fclose(arqin);
7  fclose(arqout);
```

Escrevendo dados em um arquivo texto

Exemplo de programa que faz uma cópia de um arquivo para outro, ambos informados como parâmetro do programa.

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    char aux;
    FILE *arqin, *arqout;
    if (argc < 3) {
        printf("Informe os nomes dos arquivos de entrada e
        ↪  saida.\n");
        return 1;
    }
    arqin = fopen(argv[1], "r");
    arqout = fopen(argv[2], "w");
    if (arqin == NULL || arqout == NULL) {
        printf("Problema com os arquivos.\n");
        return 1;
    }
    while (fscanf(arqin, "%c", &aux) != EOF)
        fprintf(arqout, "%c", aux);
    fclose(arqin);
    fclose(arqout);
    return 0;
```

fopen

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

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

Modos de abertura de arquivo texto

modo	operações	indicador de posição começa
r	leitura	início do arquivo
r+	leitura e escrita	início do arquivo
w	escrita	início do arquivo
w+	escrita e leitura	início do arquivo
a	(append) escrita	final do arquivo

fopen

- Se um arquivo for aberto para leitura (**r**) e ele não existir, **fopen** devolve **NULL**.
- Se um arquivo for aberto para escrita ou escrita/leitura (**w ou w+**) e existir ele é apagado e criado;
  Se o arquivo não existir um novo arquivo é criado.
  - No modo **w** você poderá fazer apenas escritas e no modo **w+** você poderá fazer tanto escritas quanto leituras.
- Se um arquivo for aberto para leitura/escrita (**r+**) e existir ele NÃO é apagado;
  Se o arquivo não existir, **fopen** devolve **NULL**.

Exemplos

Exemplo: lendo um texto na memória

- Podemos ler todo o texto de um arquivo para um vetor (deve ser grande o suficiente!) e fazer qualquer alteração que julgarmos necessária.
- O texto alterado pode então ser sobrescrito sobre o texto anterior.
- Como exemplo, vamos fazer um programa que troca toda ocorrência da letra “a” por “A” em um texto.

Lendo um texto na memória

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    if (argc < 2) {
        printf("Informe o nome do arquivo.\n");
        return 1;
    }
    FILE *arqin = fopen(argv[1], "r+");
    if (arqin == NULL) {
        printf("Problema com o arquivo.\n");
        return 1;
    }
    char aux;
    int i = 0;
    /* Determina tamanho do arquivo */
    while (fscanf(arqin, "%c", &aux) != EOF)
        i++;
    char *texto = malloc((i+1) * sizeof(char));
    ...
```

Lendo um texto na memória

```
1  int main() {
2      ...
3      rewind(arqin);
4      i = 0;
5      /* Carrega arquivo em memoria */
6      while (fscanf(arqin, "%c", &aux) != EOF) {
7          texto[i] = aux;
8          i++;
9      }
10     texto[i] = '\0';
11     /* Altera o arquivo */
12     i = 0;
13     while (texto[i] != '\0') {
14         if (texto[i] == 'a')
15             texto[i] = 'A';
16         i++;
17     }
18     rewind(arqin);
19     fprintf(arqin,"%s", texto);
20     free(texto);
21     fclose(arqin);
22     return 0;
   }
```

Resumo para se trabalhar com arquivos

- Crie um ponteiro para arquivo: **FILE *parq;**
- Abra o arquivo de modo apropriado associando-o a um ponteiro:
  - **parq = fopen(nomeArquivo, modo);** onde modo pode ser: **r, r+, w, w+**
- Leia dados do arquivo na memória.
  - **fscanf(parq, string-tipo-variavel, &variavel);**
  - Dados podem ser lidos enquanto **fscanf** não devolver **EOF**.
- Altere dados se necessário e escreva-os novamente em arquivo.
  - **fprintf(parq, string-tipo-variavel, variavel);**
- Todo arquivo aberto deve ser fechado.
  - **fclose(parq);**

Informações extras: fscanf para ler int, double, etc.

Informações extras: fscanf para int, double, etc.

- Você pode usar o **fscanf** como o **scanf** para ler dados em variáveis de outro tipo que não texto ou char.
  - Pode-se ler uma linha "1234" no arquivo texto para um **int** por exemplo:

```
1 int i;
2 fscanf(arq, "%d", &i);
```

- O mesmo vale para o **fprintf** em relação ao **printf**.
  - Neste exemplo é escrito o texto "56" no arquivo.

```
1 int i = 56;
2 fprintf(arq, "%d", i);
```

- Você pode apagar um arquivo usando a função **remove(string-nome-arq)**.

Arquivos binários

Motivação

- Vimos que existem dois tipos de arquivos: textos e binários.
- Variáveis **int** ou **float** têm tamanho fixo na memória. Por exemplo, um **int** ocupa 4 bytes.
  - Representação em texto precisa de um número variável de dígitos (`10, 5.673, 100.340`), logo de um tamanho variável.
  - Lembre-se que cada letra/dígito é um **char** e usa 1 byte de memória.
- Armazenar dados em arquivos de forma análoga a utilizada em memória permite:
  - Reduzir o tamanho do arquivo.
  - Guardar estruturas complicadas tendo acesso simples.

Arquivos binários em C

- Assim como em arquivos texto, para trabalharmos com arquivos binários devemos criar um ponteiro para arquivos.

```
FILE *nome_variavel;
```

- Podemos então associar o ponteiro com um arquivo real do computador usando o comando **fopen**.

```
FILE *arq1;
arq1 = fopen("teste.bin", "rb");
```

Abrindo um arquivo binário: fopen

Um pouco mais sobre a função **fopen()** para arquivos binários.

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

Modos de abertura de arquivo binário

modo	operações
rb	leitura
wb	escrita
r+b	leitura e escrita
w+b	escrita e leitura

Abrindo um arquivo binário: fopen

- Se um arquivo for aberto para leitura (**rb**) e não existir a função devolve **NULL**.
- Se um arquivo for aberto para escrita (**wb**) e não existir um novo arquivo é criado. Se ele existir, é sobrescrito.
- Se um arquivo for aberto para leitura/gravação (**r+b**) e existir ele NÃO é sobrescrito;
  Se o arquivo não existir a função devolve **NULL**.
- Se um arquivo for aberto para gravação/escrita (**w+b**) e existir ele é sobrescrito;
  Se o arquivo não existir um novo arquivo é criado.

Lendo e escrevendo com fread e fwrite

- As funções **fread** e **fwrite** permitem a leitura e escrita de blocos de dados.
- Devemos determinar o número de elementos a serem lidos ou gravados e o tamanho de cada um.

Lendo e escrevendo com fread e fwrite

Para escrever em um arquivo binário usamos a função **fwrite**:

```
1  size_t fwrite(void *pt-mem, size_t size, size_t
   ↪  num-items, FILE *pt-arq);
```

- **pt-mem:** Ponteiro para região da memória contendo os itens que devem ser escritos em arquivo.
- **size:** Número de bytes de um item.
- **num-items:** Número de itens que devem ser gravados.
- **pt-arq:** Ponteiro para o arquivo.

O retorno da função é o número de itens escritos corretamente.

Lendo e escrevendo com fread e fwrite

Podemos por exemplo gravar um double em formato binário como abaixo:

```c
#include <stdio.h>

int main() {
    double aux = 2.5;
    FILE *arq = fopen("teste.bin", "wb");

    if (arq == NULL) {
        printf("Erro no arquivo.\n");
        return 1;
    }

    fwrite(&aux, sizeof(double), 1, arq);
    fclose(arq);

    return 0;
}
```

Lendo e escrevendo com fread e fwrite

Para ler de um arquivo binário usamos a função **fread**:

```
1  size_t fread(void *pt-mem, size_t size, size_t
   ↪  num-items, FILE *pt-arq);
```

- **pt-mem:** Ponteiro para região da memória (já alocada) para onde os dados serão lidos.
- **size:** Número de bytes de um item a ser lido.
- **num-items:** Número de itens que devem ser lidos.
- **pt-arq:** Ponteiro para o arquivo.

O retorno da função é o número de itens lidos corretamente.

Lendo e escrevendo com fread e fwrite

Usando o exemplo anterior podemos ler um double em formato binário como segue:

```c
#include <stdio.h>

int main() {
    double aux;
    FILE *arq = fopen("teste.bin", "rb");

    if (arq == NULL) {
        printf("Erro no arquivo.\n");
        return 1;
    }

    fread(&aux, sizeof(double), 1, arq);
    printf("Lido: %lf\n", aux);

    fclose(arq);
    return 0;
}
```

Lendo e escrevendo com fread e fwrite

Podemos por exemplo gravar um vetor de doubles em formato binário:

```c
#include <stdio.h>

int main() {
    double aux[] = {2.5, 1.4, 3.6};

    FILE *arq = fopen("teste.bin", "w+b");
    if (arq == NULL) {
        printf("Erro no arquivo.\n");
        return 1;
    }

    fwrite(aux, sizeof(double), 3, arq);
    fclose(arq);
    return 0;
}
```

Lendo e escrevendo com fread e fwrite

Usando o exemplo visto, podemos ler um vetor de doubles em formato binário como segue:

```c
#include <stdio.h>
int main() {
    double aux[3];
    int i;
    FILE *arq = fopen("teste.bin", "r+b");
    if (arq == NULL) {
        printf("Erro no arquivo.\n");
        return 1;
    }

    fread(aux, sizeof(double), 3, arq);
    for (i = 0; i < 3; i++)
        printf("%lf, ", aux[i]);
    printf("\n");

    fclose(arq);
    return 0;
}
```

Lendo e escrevendo com fread e fwrite

- Lembre-se do **indicador de posição** de um arquivo, que assim que é aberto é apontado para o início do arquivo.
- Quando lemos uma determinada quantidade de itens, o indicador de posição automaticamente avança para o próximo item não lido.
- Quando escrevemos algum item, o indicador de posição automaticamente avança para a posição seguinte ao item escrito.

Lendo e escrevendo com fread e fwrite

- Se na leitura não sabemos exatamente quantos itens estão gravados, podemos usar o que é devolvido pela função **fread**:
  - Esta função devolve o número de itens corretamente lidos.
  - Se alcançarmos o final do arquivo e tentarmos ler algo, ela devolve 0.

No exemplo do vetor poderíamos ter lido os dados como segue:

```
1 double aux2;
2 while (fread(&aux2, sizeof(double), 1, arq) != 0) {
3     printf("%.2lf, ", aux2);
4 }
```

fread e fwrite

Lendo dados do arquivo:

```c
#include <stdio.h>
int main(void) {
    double aux2;
    FILE *arq = fopen("teste.bin", "r+b");
    if(arq == NULL) {
        printf("Erro");
        return 1;
    }
    while(fread(&aux2, sizeof(double), 1, arq) !=
    ↪  0) {
        printf("%.2lf, ", aux2);
    }
    printf("\n");
    fclose(arq);
    return 0;
}
```

Acesso não sequencial: fseek

- Fazemos o acesso não sequencial usando a função **fseek**.
- Esta função altera a posição de leitura/escrita no arquivo.
- O deslocamento pode ser relativo ao:
  - início do arquivo (SEEK_SET)
  - ponto atual (SEEK_CUR)
  - final do arquivo (SEEK_END)

Acesso não sequencial: fseek

```
int fseek(FILE *pt-arq, long num-bytes, int
↪  origem);
```

- **pt-arq:** ponteiro para arquivo.
- **num-bytes:** quantidade de bytes para se deslocar.
- **origem:** posição de início do deslocamento (SEEK_SET, SEEK_CUR, SEEK_END).

Por exemplo, se quisermos alterar o segundo **double** de um vetor escrito fazemos:

```
1 double aux[] = {2.5, 1.4, 3.6};
2 double aux3 = 5.0;
3 arq = fopen("teste.bin", "w+b");
4 fwrite(aux, sizeof(double), 3, arq);
5 fseek(arq, 1*sizeof(double), SEEK_SET); /* A partir do inicio
  ↪  pula um double */
6 fwrite(&aux3, sizeof(double), 1, arq);
```

fread e fwrite

Programa que escreve vetor de 3 números do tipo **double**:

```c
#include <stdio.h>

int main() {
    double aux[] = {2.5, 1.4, 3.6};

    FILE *arq = fopen("teste.bin", "w+b");
    if (arq == NULL) {
        printf("Erro no arquivo.\n");
        return 0;
    }

    fwrite(aux, sizeof(double), 3, arq);
    fclose(arq);
    return 0;
}
```

fread e fwrite

Programa que altera o arquivo:

```c
#include <stdio.h>

int main() {
    double aux = 104.98;
    FILE *arq = fopen("teste.bin", "r+b");
    if (arq == NULL) {
        printf("Erro");
        return 1;
    }

    /* seta o indicador de posição do arquivo para o início do
    ↪  segundo número */
    fseek(arq, 1*sizeof(double), SEEK_SET);
    fwrite(&aux, sizeof(double), 1, arq);
    fclose(arq);
    return 0;
}
```

Exemplo: arquivo de registros

Exemplo: arquivo de registros

- Um arquivo pode armazenar registros (como um banco de dados).
- Isso pode ser feito de forma bem fácil se lembrarmos que um registro, como qualquer variável em C, tem um tamanho fixo.
- O acesso a cada registro pode ser direto, usando a função **fseek**.
- A leitura ou escrita do registro pode ser feita usando as funções **fread** e **fwrite**.

Exemplo: arquivo de registros

Vamos fazer uma aplicação para um cadastro de alunos:

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

/* Nome do arquivo que contém o cadastro */
#define FILE_NAME "alunos.bin"

struct Aluno {
    char nome[100];
    int RA;
};
typedef struct Aluno Aluno;

/* Esta função imprime todo o conteúdo do cadastro em arquivo */
void imprimeArquivo();

/* Dado um ra passado por parâmetro, a função altera o nome da
   pessoa com este ra */
void alteraNome(int ra, char nome[]);
```

Exemplo: função que imprime arquivo

```
1  void imprimeArquivo() {
2      Aluno cadastro;
3      FILE *arq = fopen(FILE_NAME, "r+b"); /* Note que usamos r e
       ↪  não w */
4      if (arq == NULL) {
5          printf("Erro Arquivo (imprime).\n");
6          return;
7      }
8
9      printf(" ---- Imprimindo Dados ----\n");
10     while (fread(&cadastro, sizeof(Aluno), 1, arq) != 0) {
11         printf("Nome: %s,   RA: %d \n", cadastro.nome,
           ↪  cadastro.RA);
12     }
13     printf("\n");
14     fclose(arq);
15 }
```

Exemplo: função que altera um registro

```c
void alteraNome(int ra, char nome[]) {
    Aluno aluno;
    FILE *arq = fopen(FILE_NAME, "r+b");
    if (arq == NULL) {
        printf("Erro Arquivo (altera).\n");
        return;
    }

    while (fread(&aluno, sizeof(Aluno), 1, arq) != 0) {
        if (aluno.RA == ra) { /* Encontramos o Aluno */
            strcpy(aluno.nome, nome); /* Altera nome */
            fseek(arq, -1*sizeof(Aluno), SEEK_CUR); /* Volta um
            ↪  item */
            fwrite(&aluno, sizeof(Aluno), 1, arq); /*
            ↪  Sobrescreve Reg. antigo */
            break;
        }
    }
    fclose(arq);
}
```

Exemplo: função principal

```
1  int main() {
2      Aluno cadastro[] = { {"Joao", 1}, {"Batata", 2}, {"Ze", 3},
3                           {"Malu", 4}, {"Ju", 5} };
4      FILE *arq = fopen(FILE_NAME, "w+b");
5      if (arq == NULL) {
6          printf("Erro Arquivo (main).\n");
7          return 1;
8      }
9      fwrite(cadastro, sizeof(Aluno), 5, arq);
10     fclose(arq);
11
12     /* Após criado o arquivo aqui em cima, vamos alterá-lo
13        chamando a função alteraNome */
14     imprimeArquivo();
15     alteraNome(4, "Malu Mader");
16     imprimeArquivo();
17     return 0;
18 }
```

Exercícios

Exercício

- Escreva um programa que leia dois arquivos de inteiros ordenados e escreva um arquivo cuja saída é um único arquivo ordenado.
  - Vale a pena colocar o conteúdo dos arquivos de entrada em dois vetores?
  - Escreva duas versões deste programa, uma para arquivos texto e outra para arquivos binários.