Programação Procedimental

Arquivos

Aula 15

Prof. Felipe A. Louza

Roteiro

Tipos de Memória

Importante:

- Todos os **programas** vistos até agora utilizam a memória principal (RAM) para armazenar dados
- Quando o programa termina ou acaba energia, as informações são perdidas.

Tipos de Memória

Para gravar **informações** de forma *persistente*, devemos escrever em arquivos em uma **memória secundária**.

- Hard Disks, SSD, pendrive, ...

Sempre que nos referirmos a um arquivo, estamos falando de informações armazenadas em memória secundária.

Arquivos

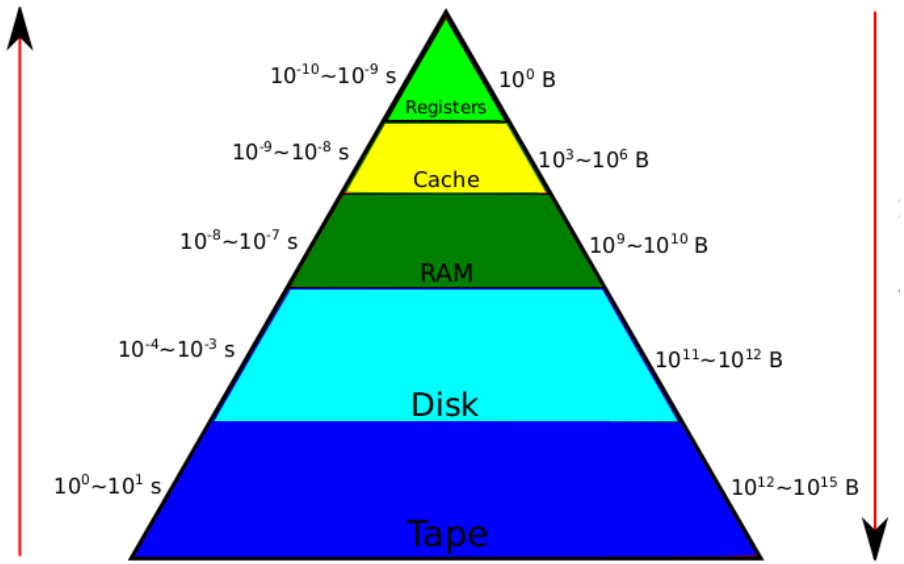
Memória secundária:

- Mantém dados de forma persistente (gravado em disco)
- Podem armazenar grande quantidade de informação
- Acesso aos dados pode ser concorrente

Tipos de Memória

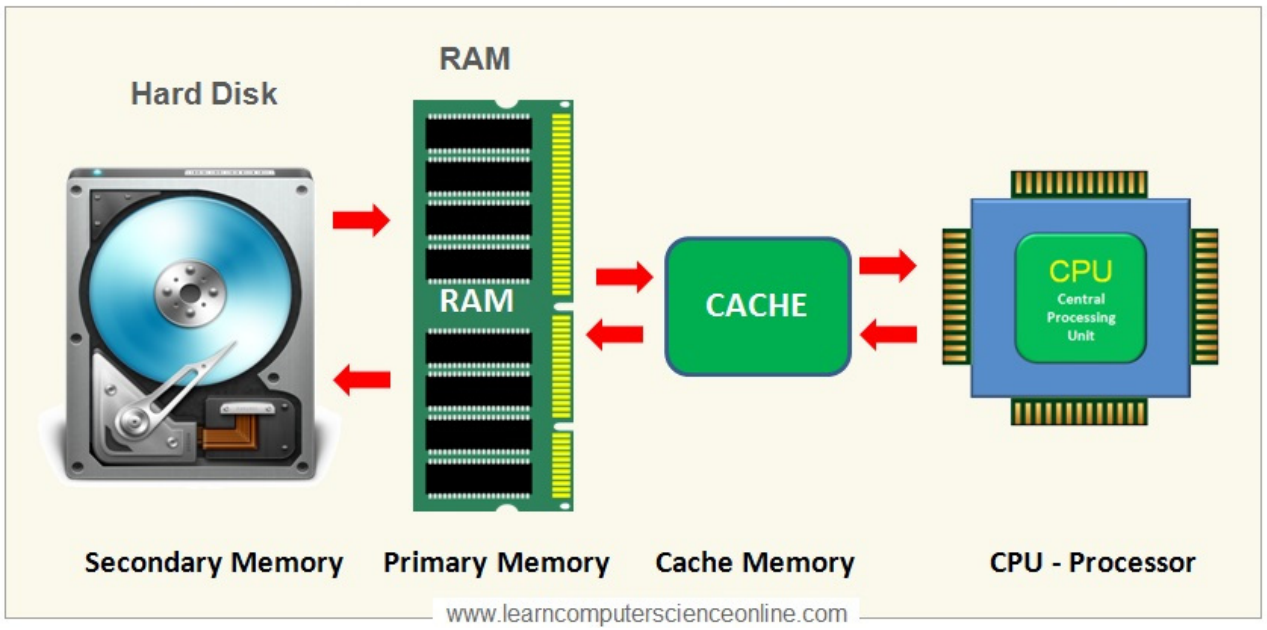
Algumas características de uma **memória secundária**:

- É muito mais lenta que a RAM.
- É muito mais barata que a memória RAM.



Organização da Memória

Organização da Memória:

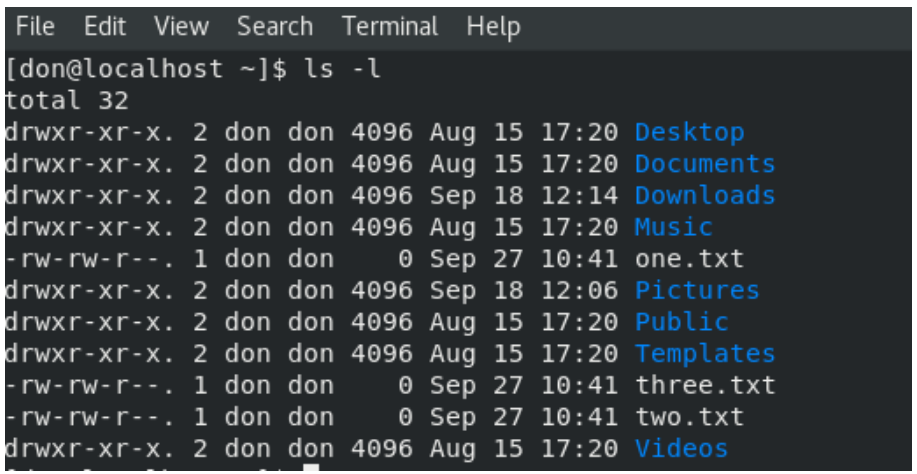


Roteiro

Arquivos

Arquivos:

- Sempre que nos referirmos a um arquivo, estamos falando de informações armazenadas em memória secundária.



Arquivos

Nomes e extensões:

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

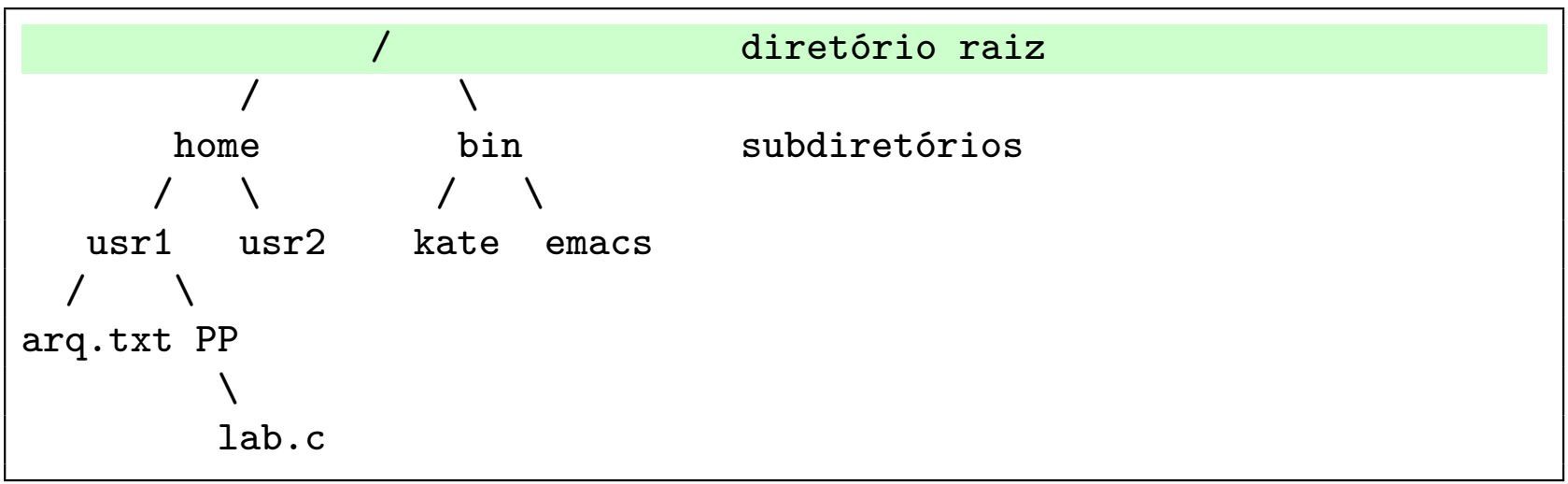
`arq.txt`	arquivo texto simples
`arq.c`	código fonte em C
`arq.pdf`	*portable document format*
`arq`*	arquivo executável (UNIX)
`arq.exe`	arquivo executável (Windows)

Figura: Algumas extensões

Diretório

Diretório:

- Também chamado de pasta.
- Contém arquivos e/ou outros diretórios.
- Uma hierarquia de diretórios no **Linux**:



Caminhos Absolutos e Relativos

O acesso a um arquivo pode ser especificado de duas formas:

- Caminho absoluto: desde o diretório raiz.

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

- Caminho relativo: a partir do diretório corrente.

```
1 arq.txt
2 PP/lab.c
```

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

Atributos de arquivos

Além do nome, arquivos possuem vários outros atributos:

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

```
1 $ ls -l
2 -rw-r--r--. 1 felipe prof 587241 Sep  8 11:46 aula09.pdf
3 -rw-r--r--. 1 felipe prof   1864 Sep  8 10:28 aula09.tex
4 -rw-r--r--. 1 felipe prof  21780 Sep  8 11:46 c-arquivos.tex
```

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

Permissão de acesso

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

```
1 $ ls -l
2 -rw-r----- 1 felipe prof  545 Nov  8  2005 arq.c
3 drwxr-xr-x 2 felipe prof 4096 Jun  6 14:54 exemplos/
```

Tipos de permissão:

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

Tipos de arquivos

Do ponto de vista do programador existem apenas dois tipos de arquivo:

1. **Arquivo texto**: Armazena caracteres seguindo uma codificação (UTF-8, por exemplo)

```
1 "The quick brown fox jumps over the lazy dog" is an English-language
2 pangram a sentence that contains all of the letters of the alphabet.
3 ...
```

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

Roteiro

Arquivos texto em C

Arquivos em C:

- Para trabalhar com arquivos usamos um ponteiro especial: **um ponteiro para arquivos.**
- Tipo `FILE`:

```c
#include <stdio.h>
...
//ponteiro para um arquivo
FILE *arq;
```

https://www.geeksforgeeks.org/data-type-file-c/

Arquivos texto em C

Para abrir um arquivo, utilizamos a função `fopen()`:

```
1 FILE * fopen(const char *filename, const char *mode);
```

- Parâmetros:
  1. `filename`: nome do arquivo a ser aberto
  2. `mode`: modo de abertura do arquivo

- Exemplo:

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

fopen()

Sobre os parâmetros:

1. `filename`: pode ser o nome absoluto ou relativo do arquivo
2. `mode`:

Modo	Operações	Observações
`"r"`	leitura	
`"r+"`	leitura e escrita	(arquivo tem que existir)
`"w"`	escrita	(sobrescreve o arquivo, se existir)
`"w+"`	escrita e leitura	
`"a"`	escrita	(acrescenta dados no fim do arquivo)

Figura: Modos de abertura de arquivo texto

`"b"` indica "modo" binário

Abrindo um Arquivo Texto

Sempre antes de acessar um arquivo, devemos abri-lo com a função `fopen()`.

- A função retorna um endereço para um ponteiro (`FILE *`), em caso de erro a função retorna `NULL`.

```
1 FILE *arq;
2 arq = fopen("teste.txt","r");
3 if(arq == NULL)
4   perror("Erro ao abrir o arquivo.\n");
5 else
6   printf("Arquivo aberto para leitura.\n");
7 ...
```

A função `perror()` obtém e exibe uma mensagem explicativa.

Fechando um Arquivo Texto

Precisamos também fechar um arquivo que foi aberto:

```
1 int fclose(FILE* arq);
```

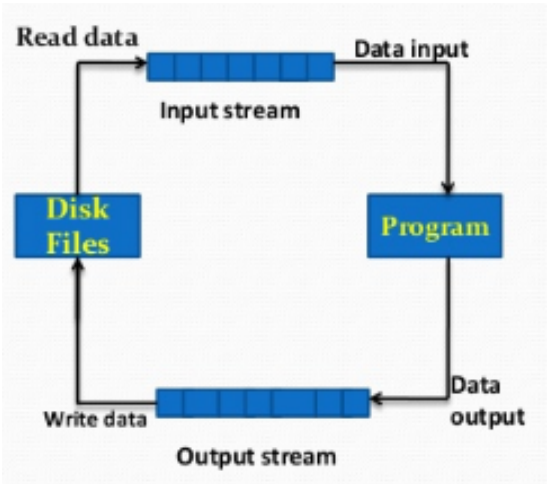
- A função retorna 0 em caso de sucesso e EOF (-1) caso contrário.
- Importante:
  - Quando **escrevemos** em um arquivo, `fclose()` garante que os dados serão efetivamente escritos no arquivo.

```
1 FILE *arq;
2 arq = fopen("teste.txt","r");
3 ...
4 fclose(arq);
```

Fechando um Arquivo Texto

Normalmente existe um buffer de escrita e um de leitura de arquivos.

- Aumentam a eficiência das operações, pois reduzem o número de acessos ao disco.
- O conteúdo do arquivo em disco pode estar desatualizado em relação às modificações feitas por um programa.



Roteiro

Escrevendo em um Arquivo Texto

Escrevendo em um arquivo:

- Primeiro, precisamos abrir o arquivo de forma apropriada:

```
1 FILE *arq = fopen("teste.txt","w"); //ou "w+" ou "a"
```

- Podemos usar a função `fprintf()`, semelhante ao `printf()`:

```
1 int fprintf(FILE *arq, const char *format, ... );
```

  - Precisamos passar o ponteiro para o arquivo.

Escrevendo em um Arquivo Texto

Exemplo 1:

```c
#include <stdio.h>

int main(){

  FILE *arq = fopen("texto.txt", "w");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  // escrevendo em um arquivo texto
  fprintf(arq, "Hello FILE!!\n");

  fclose(arq);

return 0;
}
```

Escrevendo em um Arquivo Texto

Exemplo 2:

```c
#include <stdio.h>

int main(){

  FILE *arq = fopen("numeros.txt", "w");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  // escrevendo em um arquivo texto
  int i;
  for(i=0; i<10; i++)
    fprintf(arq, "%d\n", i);

  fclose(arq);

return 0;
}
```

Escrevendo em um Arquivo Texto

Outras funções:

- `fputs()`: escreve uma string no arquivo

```
1 int fputs(const char *str, FILE *arq);
```

- `fputc()`: escreve um caractere no arquivo

```
1 int fputc(int character, FILE *stream);
```

  - As funções retornam -1 em caso de erro.

Roteiro

Arquivos texto em C

Relembrando:

- Para trabalhar com arquivos usamos **um ponteiro para arquivos.**
- Sempre antes de acessar um arquivo, devemos abri-lo com a função `fopen()`:

```
1 FILE *arq = fopen("teste.txt","r");
2 if(arq==NULL){
3   perror("Erro ao abrir o arquivo");
4   return 1;
5 }
6 ...
7 fclose(arq);
```

- Precisamos também fechar um arquivo que foi aberto.

fopen()

Modos de acesso:

Modo	Operações	Observações
`"r"`	leitura	
`"r+"`	leitura e escrita	(arquivo tem que existir)
`"w"`	escrita	(sobrescreve o arquivo, se existir)
`"w+"`	escrita e leitura	
`"a"`	escrita	(acrescenta dados no fim do arquivo)

Figura: Modos de abertura de arquivo texto

`"b"` indica "modo" binário

Roteiro

Lendo Dados de um Arquivo Texto

Lendo um arquivo:

- Primeiro, precisamos abrir o arquivo de forma apropriada:

```c
FILE *arq = fopen("teste.txt","r"); //ou "r+" ou "w+"
```

- Podemos usar a função `fscanf()`, semelhante ao `scanf()`:

```c
int fscanf(FILE *arq, const char *format, ... );
```

  - A função retorna o número de argumentos lidos ou -1 (`EOF`) se o fim do arquivo for atingido

Lendo Dados de um Arquivo Texto

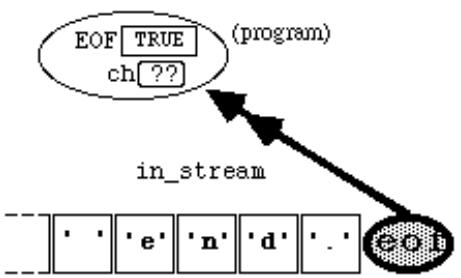
Exemplo 3:

```c
#include <stdio.h>

int main(){

  FILE *arq = fopen("texto.txt", "r");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  char texto[100];
  // lende de um arquivo texto
  fscanf(arq, "%s", texto);
  printf("%s", texto);

  fclose(arq);

return 0;
}
```

Lendo Dados de um Arquivo Texto

Fim de arquivo (`EOF`):

- Quando um arquivo é aberto, um **indicador de posição** no arquivo é criado, e recebe o início do arquivo.
- Para cada dado lido do arquivo, este indicador de posição é automaticamente incrementado.
- Quando o indicador chega ao fim do arquivo:
  - A função `fscanf()` devolve `EOF` (*End Of File*)



Lendo Dados de um Arquivo Texto

Para ler todos os dados de um arquivo texto:

- Podemos usar um laço que será executado enquanto não chegarmos no fim do arquivo.

```
1 char aux;
2 FILE *arq = fopen ("teste.txt", "r");
3 while (fscanf(arq, "%c", &aux) != EOF){
4   printf("%c", aux);
5 }
6 fclose(arq);
```

Lendo Dados de um Arquivo Texto

Outras funções:

- `fgets()`: lê uma linha, incluindo o '\n' de um arquivo

```
1 char *fgets (char *str, int num, FILE *arq);
```

- `fgetc()`: lê um caractere e retorna como um inteiro

```
1 int fgetc(FILE *arq);
```

  - As funções retornam `NULL` em caso de erro.

- `feof()`: retorna 0 se a posição atual não for o fim do arquivo

```
1 int feof(FILE *arq);
```

Roteiro

Lendo Dados de um Arquivo Texto

Podemos voltar ao início usando a função `rewind()`:

```
1  ...
2  while(fscanf(arq,"%c",&aux) != EOF){
3    printf("%c",aux);
4  }
5
6  printf{"\n\n -----Imprimindo novamente\n\n");
7  rewind(arq);
8
9  while(fscanf(arq,"%c",&aux) != EOF){
10   printf("%c",aux);
11 }
12 ...
```

Lendo Dados de um Arquivo Texto

Tamanho de um aquivo:

- Para descobrir o tamanho de um arquivo podemos mover o **indicador de posição** para o fim do arquivo:

```
1 fseek(arq, 0, SEEK_END);
```

- Em seguida, a função `ftell()` retorna a posição atual do **indicador de posição** ou -1 em caso de erro

```
1 size_t size = ftell(arq);
2 rewind(arq);
```

Roteiro

Tipos de arquivos

Vimos que existem dois tipos de arquivos:

1. **Arquivo texto**:

```
1 "The quick brown fox jumps over the lazy dog" is an English-language
2 pangram a sentence that contains all of the letters of the alphabet.
3 ...
```

2. **Arquivo binário**: Sequência de bits sujeita às convenções dos programas que o gerou, não legíveis diretamente.

```
1 $ less a.out
2 "a.out" may be a binary file.  See it anyway?
```

```
1 ^?ELF^B^A^A^@^@^@^@^@^@^@^@^B^@>^@^A^@^@^@<80>^P@^@^@^@^@@^@^
2 @^@^@^@^@^@4 @^@^@^@^@^@\^@^@^@^@^@^@\^@^@^@^@^@^@^D^@<E5>td
3 ...
```

Motivação

Vamos pensar na seguinte situação:

- Para representar um número (`int`) como texto, vamos gastar um número variável de bytes:
  - 10 (`2 bytes`)
  - 1.000 (`4 bytes`)
  - 100.000 (`6 bytes`)
  - ...
  - 2.147.483.648 (`10 bytes`)

- Com arquivos binários podemos armazenar dados em arquivos de **forma análoga a utilizada na RAM** (`4 bytes` para um `int`) .

Arquivos binários em C

Relembrando:

- Para trabalhar com arquivos usamos **um ponteiro para arquivos.**
- Antes de acessar um arquivo, devemos abri-lo com a função `fopen()`:

```
1 FILE *arq;
2 arq = fopen("teste.txt", "rb");
3 if(arq==NULL){
4   perror("Erro ao abrir o arquivo");
5   return 1;
6 }
7 ...
8 fclose(arq);
```

- Precisamos também fechar um arquivo que foi aberto.

fopen()

Modos de acesso:

Modo	Operações	Observações
`"rb"`	leitura	
`"rb+"`	leitura e escrita	(arquivo tem que existir)
`"wb"`	escrita	(sobrescreve o arquivo, se existir)
`"wb+"`	escrita e leitura	
`"ab"`	escrita	(acrescenta dados no fim do arquivo)

Figura: Modos de abertura de arquivo texto

`"b"` indica "modo" binário

Roteiro

Escrevendo Dados de um Arquivo Binário

Escrevendo em um arquivo:

- Primeiro, precisamos abrir o arquivo de forma apropriada:

```c
FILE *arq = fopen("teste.bin","wb"); //ou "wb+" ou "ab"
```

- Podemos usar a função `fwrite()`:

```c
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *arq);
```

  - A função retorna o número de elementos gravados com sucesso

Escrevendo Dados de um Arquivo Binário

Exemplo 1:

```c
#include <stdio.h>

int main(){

  FILE *arq = fopen("teste.bin", "wb");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  // escrevendo sizeof(int) bytes no arquivo
  int inteiro = 10;
  fwrite(&inteiro, sizeof(int), 1, arq);

  fclose(arq);

return 0;
}
```

Escrevendo Dados de um Arquivo Binário

Exemplo 2:

```c
#include <stdio.h>

int main(){

  FILE *arq = fopen("teste.bin", "wb");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  int inteiro = 10;
  // escrevendo sizeof(int) * 1 bytes no arquivo
  fwrite(&inteiro, sizeof(int), 1, arq);

  char palavra[10] = "Palavra";
  // escrevendo sizeof(char) * 10 bytes no arquivo
  fwrite(palavra, sizeof(char), 10, arq);
  fclose(arq);

return 0;
}
```

Escrevendo Dados de um Arquivo Binário

Exemplo 3:

```c
#include <stdio.h>

int main(){

  FILE *arq = fopen("vetor.bin", "wb");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  int n = 10;
  int vetor[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

  // armazena o tamanho do vetor
  fwrite(&n, sizeof(int), 1, arq);
  // escrevendo todos os elementos do vetor (sizeof(int) * n bytes)
  fwrite(vetor, sizeof(int), n, arq);
  fclose(arq);

return 0;
}
```

Lendo Dados de um Arquivo Binário

Lendo um arquivo:

- Primeiro, precisamos abrir o arquivo de forma apropriada:

```c
FILE *arq = fopen("teste.bin","rb"); //ou "rb+" ou "wb+"
```

- Podemos usar a função `fread()`:

```c
size_t fread(void *ptr, size_t size, size_t count, FILE *arq);
```

  - A função retorna o número de elementos lidos

Lendo Dados de um Arquivo Binário

Exemplo 4:

```c
#include <stdio.h>

int main(){

  FILE *arq = fopen("teste.bin", "rb");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  int inteiro;
  // lendo um inteiro
  fread(&inteiro, sizeof(int), 1, arquivo);
  char caractere;
  // lendo um caractere
  fread(&caractere, sizeof(char), 1, arquivo);

  fclose(arquivo);

return 0;
}
```

Lendo Dados de um Arquivo Binário

Exemplo 5:

```c
#include <stdio.h>

int main(){

  FILE *arq = fopen("vetor.bin", "rb");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  // lendo o tamanho do vetor
  int n;
  fread(&n, sizeof(int), 1, arq);

  ...
```

Lendo Dados de um Arquivo Binário

Exemplo 5: (continuação)

```
1   ...
2
3   // alocando e lendo o vetor (ou seja, sizeof(int) * n bytes)
4   int *vetor = malloc(n * sizeof(int));
5   fread(vetor, sizeof(int), n, arq);
6
7   // imprimindo dados lidos:
8   for (int i = 0; i < n; i++)
9     printf("%d ", vetor[i]);
10
11  // liberando memória e fechando o arquivo
12  free(vetor);
13  fclose(arq);
14
15 return 0;
16 }
```

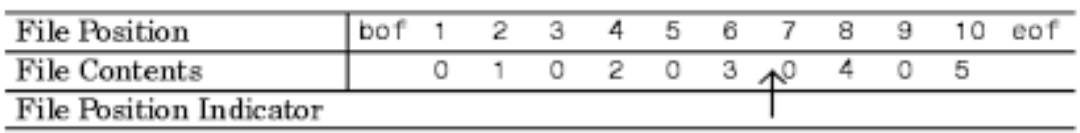
Leitura e Escrita

Importante:

- Lembre-se do **indicador de posição** de um arquivo, que assim que é aberto é **apontado** para o início do arquivo
- Quando lemos/escrevemos **itens**, o indicador de posição automaticamente avança para o próximo item não lido.

```
1 fread(ptr, sizeof(int), count, arq);
```

  - Quando a função `fread()` devolve um valor diferente de `count`, chegamos no `EOF` (*End Of File*)



Lendo Dados de um Arquivo Binário

No exemplo do vetor poderíamos ter lido os dados da seguinte forma:

```
1  ...
2  FILE *arq = fopen("vetor.dat", "rb");
3
4  int n;
5  while(fread(&n, sizeof(int), 1, arq)!=0){
6      printf("%d\n", n);
7  }
8
9  fclose(arq);
10 ...
```

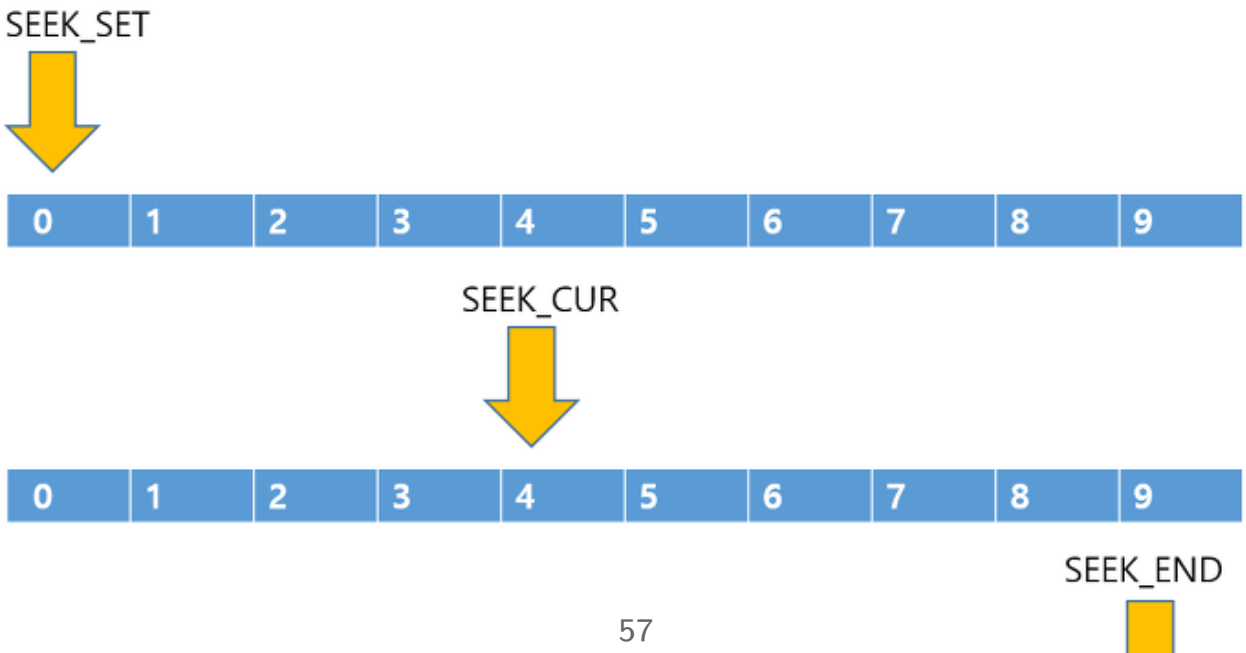
Roteiro

Acesso Não Sequencial

Podemos acessar um arquivo de forma não sequencial:

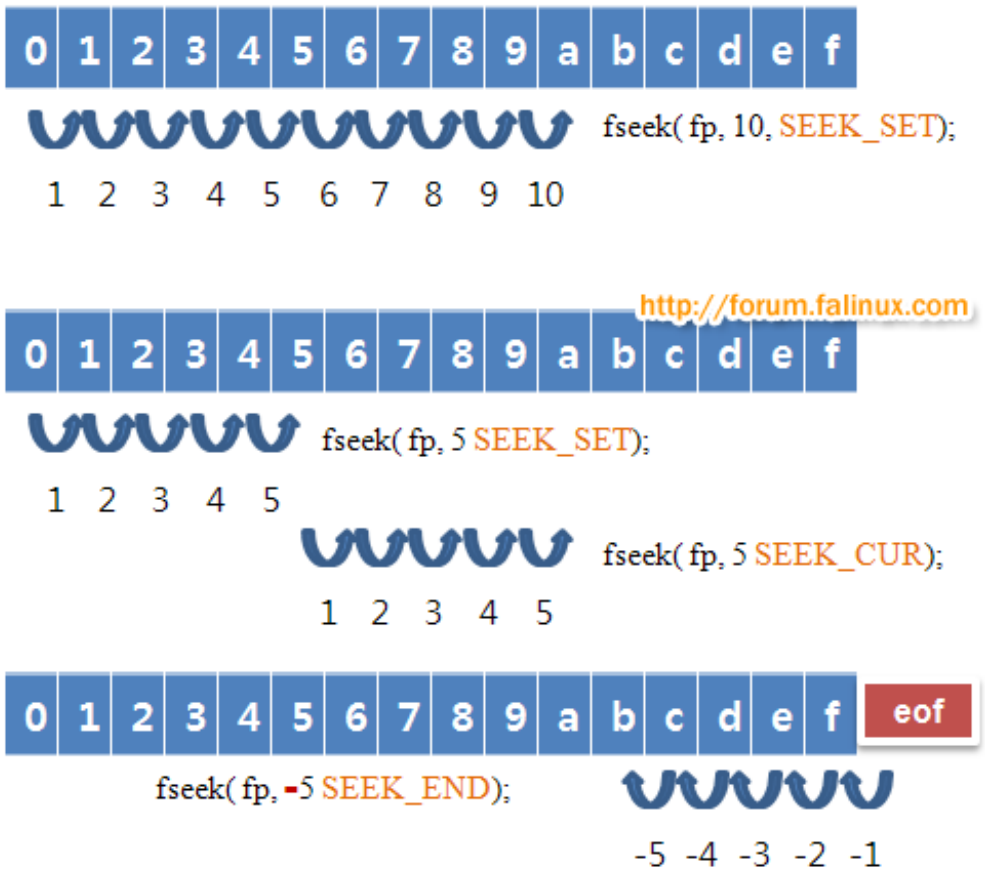
```
1 int fseek(FILE *arq, long int Nbytes, int origem);
```

- Esta função altera a posição de leitura/escrita no arquivo.
- O deslocamento (`Nbytes`) pode ser relativo ao:
  1. início do arquivo (`SEEK_SET`)
  2. ponto atual (`SEEK_CUR`)
  3. final do arquivo (`SEEK_END`)



Acesso Não Sequencial

Acesso não sequencial:



Acesso Não Sequencial

Exemplo 1: vamos alterar o **3º elemento** de um vetor já escrito:

```c
#include <stdio.h>

int main(){

  arq = fopen("teste.bin", "w+b");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  double aux[]={2.5, 1.4, 3.6};
  fwrite(aux, sizeof(double), 3, arq);

  double novo=5.0;
  fseek(arq, 2*sizeof(double), SEEK_SET); //pula 2 doubles
  fwrite(&novo, sizeof(double), 1, arq);

  fclose(arq);

return 0;
}
```

Acesso Não Sequencial

A função `rewind` (re)posiciona o indicador de posição no início do arquivo

```
1 void rewind(FILE *arq);
```

- A função não retorna nenhum valor;

Acesso Não Sequencial

Exemplo 2: vamos alterar o **1º elemento** de um vetor já escrito:

```c
#include <stdio.h>

int main(){

  arq = fopen("teste.bin", "w+b");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  double aux[]={2.5, 1.4, 3.6};
  fwrite(aux, sizeof(double), 3, arq);

  double novo=5.0;
  rewind(arq); // equivalente a fseek(arq, 0, SEEK_SET);
  fwrite(&novo, sizeof(double), 1, arq);
  fclose(arq);

return 0;
}
```

Acesso Não Sequencial

A função `ftell` retorna a posição atual em um arquivo (em bytes):

```
1 long int ftell(FILE *arq);
```

- Se ocorre um erro, a função retorna `-1`

Acesso Não Sequencial

Exemplo 3:

```c
#include <stdio.h>

int main(){

  arq = fopen("teste.bin", "wb");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  int i, v[5] = {1, 3, 5, 7, 9};
  for(i=0; i<5; i++){ // escrevendo no arquivo
    printf("%ld\n", ftell(arq));
    fwrite(&v[i], sizeof(int), 1, arq);
  }
  fclose(arq);

return 0;
}
```

Relembrando: quando escremos o indicador de posição avança automaticamente

Acesso Não Sequencial

Como descobrir o tamanho de um arquivo?

1. Vamos mover o indicador de posição para o final do arquivos

```
1 fseek(arq, 0, SEEK_END);
```

2. Com a função `ftell()`, descobrimos o número de bytes

```
1 size_t tamanho = ftell(arq);
2 rewind(arq);
```

Acesso Não Sequencial

Exemplo 4: descobrindo o tamanho de um arquivo:

```c
#include <stdio.h>

int main(int argc, char *argv[]) {

  arq = fopen(argv[1], "rb");
  if(arq == NULL){
    perror("Erro ao abrir o arquivo.\n");
    return 1;
  }

  // move para o indicador de posição para o final do arquivo
  fseek(arq, 0, SEEK_END);
  size_t tamanho = ftell(arq); // captura a posição atual (em bytes)
  rewind(arq);

  printf("O arquivo %s tem %ld bytes\n", argv[1], tamanho);
  fclose(arq);

return 0;
}
```

Acesso Não Sequencial

As funções `fseek()`, `rewind()` `ftell()` também podem ser utilizados em arquivos de texto.

Fim

Dúvidas?

Roteiro

Referências

1. Materiais adaptados dos slides do Prof. Eduardo C. Xavier, da UNICAMP.
2. Materiais adaptados dos slides do Prof. Túlio Toffolo, da UFOP.