Operações Básicas em Arquivos

6897/9895 – Organização e Recuperação de Dados
Profa. Valéria D. Feltrim

UEM – CTC – DIN

Slides preparados com base no Cap. 2 do livro FOLK, M.J. & ZOELLICK, B. *File Structures*. 2nd Edition, Addison-Wesley Publishing Company, 1992, e no material disponível em http://www.icmc.usp.br/~sce183

Diferentes “visões” do arquivo

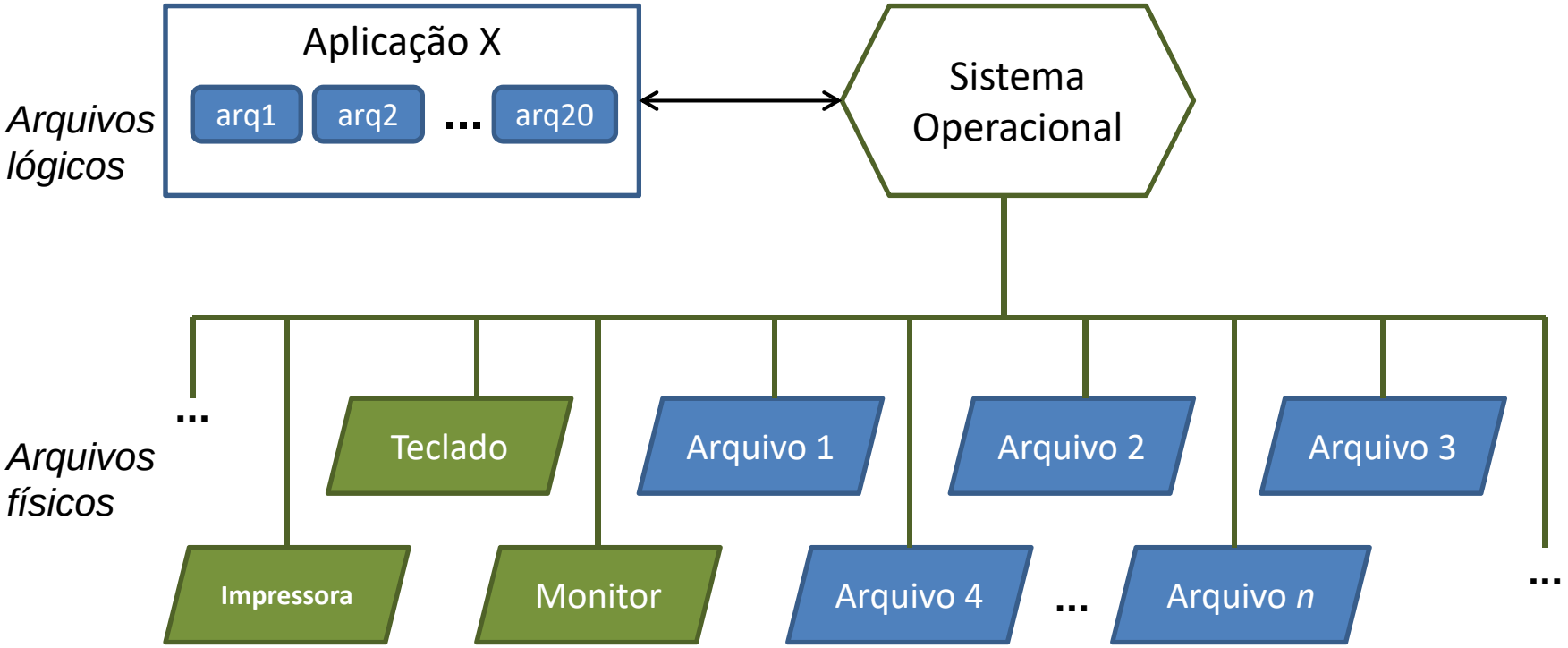
- Arquivo físico e arquivo lógico
  - Arquivo físico: quando nos referimos a um certo arquivo em disco, estamos nos referindo a uma sequência particular de bytes armazenados no disco
    - Um disco pode conter centenas ou milhares de arquivos físicos distintos
  - Arquivo lógico: é o arquivo do ponto de vista da aplicação que o acessa
    - Cada arquivo manipulado pela aplicação é tratado como um canal de comunicação estabelecido entre a aplicação e o arquivo físico
    - Podemos usar como analogia uma linha conectada a uma rede telefônica

Diferentes “visões” do arquivo

- Apesar de um disco poder conter milhares de arquivos físicos, um aplicativo é, em geral, limitado a utilizar um número pré-definido de arquivos lógicos simultaneamente
- É responsabilidade do S.O. tomar conta desse “sistema” que estabelece os canais de comunicação entre aplicativos e arquivos físicos
- Um arquivo lógico pode estar associado a um arquivo físico em disco ou a outros dispositivos de E/S, como o teclado (*in*) e o vídeo (*out*)
- Antes que o aplicativo possa abrir um arquivo, o S.O. precisa ser informado sobre o modo como a associação entre os arquivos lógico e físico deve ser feita
  - A maneira de fazer isso varia de acordo com o S.O. e a linguagem de programação utilizada

Associação do arquivo lógico ao físico

- A aplicação solicita que um nome lógico seja associado a um arquivo físico específico e o S.O. estabelece a associação



Operações em arquivos

- Associação do arquivo lógico ao físico
  - Em Turbo **Pascal**

```
var  arq: file of reg;
assign(arq,'meuarq.dat')
```

  - Em **C**, a associação é feita no momento da abertura do arquivo

```
FILE *parq;
if (( parq = fopen ("meuarq.dat","r" )) == NULL)
   printf("Erro...");
```

Criar e abrir arquivos

- Uma vez feita a associação, em geral, temos duas opções:
  - Abrir um arquivo já existente
    - O arquivo fica pronto para uso pelo aplicativo
    - O ponteiro de L/E é posicionado no início do arquivo e o aplicativo pode começar a ler/escrever
    - O conteúdo do arquivo não é afetado pelo comando de abertura (**dependendo do modo de abertura**)
  - Criar um arquivo novo
    - A criação de um arquivo também deixa o arquivo pronto para uso, mas a única operação que faz sentido é a de escrita

Criar e abrir arquivos em Pascal

- Em **Pascal**:
  - Primeiramente é feita a associação de nomes físico e lógico (comando "assign")
  - Abrir arquivos já existentes

    assign (arq,"meuarq.dat");

    reset (arq);
  - Criação de arquivos novos (se existir, apaga)

    assign (arq,"meuarq.dat");

    rewrite (arq);

Criar e abrir arquivos em C

- Em **C**:
  - A associação nome físico/lógico é feita no momento da abertura/criação
  - Comando open
    - Se o arquivo existe, abre; senão, cria
    - O modo de abertura é especificado por meio de parâmetros
    - open() é um comando baseado no UNIX (chamada de sistema) e fopen() é uma função da biblioteca *stdio.h*
  - Os exemplos a seguir utilizam as chamadas de sistema open(), close(), read(), write() em vez das funções da biblioteca stdio.h por serem comandos mais próximos do sistema de I/O, permitindo um melhor entendimento da interface com o S.O.

Criar e abrir arquivos em C

- Em C:

fd = open(filename, flags [, pmode]);

  - fd (*file descriptor):* descritor do arquivo, usado como identificação do arquivo lógico – também chamado de *file handler*
    - É um valor inteiro
    - No caso de erro na abertura do arquivo, o valor retornado por *open* é negativo (em geral, -1)
  - *filename*: *string* contendo o nome físico do arquivo
  - flags: valor inteiro que controla o modo de abertura
  - pmode: número inteiro octal que indica permissões e só é obrigatório quando a *flag* O_CREAT é especificada
  - Utiliza a biblioteca *fcntl.h*

Criar e abrir arquivos em C

- Alguns valores de *flag* pré-definidos na *fcntl.h*:
  - O_APPEND: abre para escrita no final do arquivo
  - O_CREAT: cria e abre o arquivo para escrita; sem efeito se o arquivo já existe
  - O_EXCL: retorna erro se O_CREAT foi especificado e o arquivo já existe
  - O_RDONLY: abre para leitura apenas
  - O_WRONLY: abre para escrita apenas
  - O_RDWR: abre para leitura e escrita
  - O_TRUNC: se o arquivo existe, “trunca” o arquivo para tamanho nulo destruindo seu conteúdo
- Outros valores de *flag* e as regras de uso das mesmas estão em livros ou manuais da linguagem C

Criar e abrir arquivos em C

- pmode é um número inteiro octal que indica:
  - Como o arquivo pode ser utilizado (leitura (r), escrita (w), execução (x))
  - E por quem (dono, grupo, todos)
    - (*owner/group/world)* (r w x/ r w x/ r w x)

Herança do Unix:
O pmode está mais ligado ao S.O. do que a LP

- Exemplo:
  - pmode = 0751 permite
    - Execução por todos
    - Execução e leitura pelo grupo
    - Execução, leitura e escrita pelo dono do arquivo

r	w	x	r	w	x	r	w	x
1	1	1	1	0	1	0	0	1
owner			*group*			*world*		

pmode **0777** libera todas as permissões

O zero antes do número 751 indica que o numeral a seguir é um octal

Criar e abrir arquivos em C

- Exemplos de uso do comando open:
  - fd = open(filename, O_RDWR | O_CREAT, 0751);
    - Abre o arquivo existente para L/E ou cria um novo se necessário
    - Se o arquivo existe, ele é aberto sem ser modificado
    - Posiciona o ponteiro de leitura/escrita (L/E) no início do arquivo
  - fd = open(filename, O_RDWR|O_CREAT|O_TRUNC, 0751);
    - Idem ao acima, mas se o arquivo já existe, seu conteúdo é truncado (tamanho passa a ser zero)
  - fd = open(filename, O_WRONLY|O_CREAT|O_EXCL, 0751);
    - Cria apenas se arquivo não existe, caso contrário devolve erro (valor negativo)
    - Se cria, abre o arquivo para escrita apenas

Criar e abrir arquivos em C

- As funções da linguagem C para manipulação de arquivos fazem parte da biblioteca *stdio.h*
- São funções com um nível maior de abstração do que os comandos incorporados do Unix
- O descritor de arquivo usado por elas é um ponteiro do tipo FILE, que é uma estrutura própria do C para armazenamento do descritor
- Exemplo do uso do comando fopen:

```
FILE * pFile;
pFile = fopen ("myfile.txt","w");
```

  - modo de abertura → "r" (read), "w" (write), "a" (append)
    "r+", "w+", "a+", "rb", "wb", "ab", "r+b", "w+b", "a+b“

- Mais infos: http://www.cplusplus.com/reference/clibrary/cstdio/fopen/

Fechamento de arquivos

- Fechamento de arquivos
  - Encerra a associação entre o arquivo lógico e o físico
  - O descritor fica disponível para uso por outro arquivo
  - Garante que todas as informações do arquivo foram atualizadas (no caso da escrita ser feita em blocos, o fechamento garante que o conteúdo dos *buffers* foi enviado para o disco)
  - O S.O. normalmente fecha qualquer arquivo que ficou aberto quando um programa termina
    - O fechamento explícito é necessário como:
      - Uma prevenção contra interrupções
      - Para liberação das estruturas associadas aos descritores para novos arquivos

Fechamento de arquivos em C

- Exemplos em C:

```
int fd;
fd = open(filename, O_RDONLY);
close(fd);
```

```
FILE *fd;
fd = fopen(nomearq, "r");
fclose(fd);
```

- Perceba que o tipo de descritor de arquivo muda da função "open" para "fopen"
- A função "fopen" abstrai os detalhes do comando "open", que é de mais baixo nível, assim como todas as funções "f*" para manipulação de arquivos
- A operação de fechamento tem que ser coerente com a operação de abertura (open/close) → (fopen/fclose)

Leitura e escrita em arquivos

- Leitura e escrita em arquivos
  - Funções genéricas de baixo nível

read(source_file, destination_addr, size)

- source_file: descritor de arquivo obtido na abertura
- destination_addr: local onde a informação obtida do arquivo deve ser armazenada → o endereço inicial de um bloco de memória
- size: número de bytes a serem lidos do arquivo

write(destination_file, source_addr, size)

- os parâmetros são análogos, apenas operam na direção inversa

Leitura e escrita em arquivos em C

- Leitura em arquivos em C
  - Protótipo da função

    int read (**int** *fd*, **void** **buf*, **int** *nbyte*);
    - Retorno: um valor **inteiro** correspondente à **quantidade de bytes lidos**
    - Três parâmetros:
      - **int *fd***: *file descriptor* → inteiro retornado pela função open() que identifica o arquivo do qual se vai ler
      - **void ***buf***: *buffer* → ponteiro *void* para a variável que armazenará os bytes lidos
        - » É um tipo "nulo" que é compatível com todos os outros tipos

          Uma função cujo tipo é *void* não retorna nada

          Um ponteiro *void* pode apontar qualquer coisa
      - **int *nbyte***: *number of bytes*→ número de bytes a serem lidos do *fd*

Leitura e escrita em arquivos em C

- Funções de leitura da **stdio.h** que funcionam se o arquivo foi aberto com *fopen*
  - **int** fgetc **(FILE *** *fd***)**
    - Retorna o caracter corrente apontado pelo ponteiro de L/E do arquivo *fd*
    - Equivalente a getc()
  - **char *** fgets **(char *** *str***, int** *num***, FILE *** *fd* **)**
    - Lê caracteres de *fd* até *num-1* ou até encontrar uma quebra de linha ou o fim do arquivo, e os armazena em *str* como uma string C
    - O caracter "nulo" ('\0'), que finaliza strings em C, é automaticamente anexado ao final da string lida
  - **int** fread **(void *** *ptr***, int** *size***, int** *count***, FILE *** *fd* **)**
    - Lê um bloco de bytes de *fd* a partir da posição atual do ponteiro de L/E
    - Lê um vetor de *count* elementos, no qual cada elemento tem *size* bytes de tamanho (uso da função sizeof()), e o armazena no bloco de memória apontado por *ptr*
    - A quantidade de bytes lidos é dada por *size * count*
    - Retorna o número de elementos lidos com sucesso
- Mais infos: http://www.cplusplus.com/reference/clibrary/cstdio/

Leitura e escrita em arquivos em C

- Escrita em arquivos em C
  - Protótipo da função

    int write (**int** *fd*, **void** **buf*, **int** *nbyte*);

    - Retorno: um valor **inteiro** → o número de bytes que foi escrito ou -1 se ocorreu erro
    - Três parâmetros:
      - **int *fd***: *file descriptor* → inteiro retornado pela função open() que identifica o arquivo do qual se vai escrever
      - **void ***buf***: *buffer* → ponteiro *void* para a var que armazena os bytes a serem escritos em *fd*
      - **int *nbyte***: *number of bytes*→ número de bytes a serem escritos do *fd*

Leitura e escrita em arquivos em C

- Funções de escrita da **stdio.h** que funcionam se o arquivo foi aberto com *fopen*
  - **int** fputc **(int** character**, FILE *** *fd* **)**
    - Escreve um caracter no arquivo *fd* e avança o ponteiro de leitura/escrita do arquivo
    - Equivalente a putc()
  - **int** fputs **(const char *** *str***, FILE *** *fd* **)**
    - Escreve a string apontada por *str* no arquivo *fd*
    - O caracter nulo ('\0') não é copiado para o arquivo
  - **int** fwrite **(const void *** *ptr***, int** *size***, int** *count***, FILE *** *fd* **)**
    - Escreve um bloco de dados em *fd* a partir da posição atual do ponteiro de L/E
    - Escreve um vetor de *count* elementos, sendo cada elemento do tamanho de *size* bytes (uso da função sizeof()), a partir do bloco de memória apontado por *ptr*
    - A quantidade de bytes escritos é dada por *size * count*
    - Retorna o número de elementos escritos com sucesso
- Mais infos: http://www.cplusplus.com/reference/clibrary/cstdio/

Exemplo leitura/escrita em C

```
/* list.c - programa que lê caracteres de
      um arquivo e os escreve na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
       fputc (c, stdout);
   fclose (fd);
}
```

Relembrando: C é *case sensitive!*

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

*Inclusão de **bibliotecas** contendo definições de constantes e protótipos de funções*

stdio.h → ***St**andar**d** **I**nput and **O**utput Library*

fcntl.h → ***F**ile **C**ontrol Library*
(define constantes e parâmetros usados pela função **open**)

#define STDOUT 1

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
      fputc (c, stdout);
   fclose (fd);
}
```

#define EOF -1

Este **stdout** é um FILE*

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

Definição de variáveis: pode ser feita em qualquer ponto do programa

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
       fputc (c, stdout);
   fclose (fd);
}
```

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

Descritor inteiro

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
       fputc (c, stdout);
   fclose (fd);
}
```

Descritor ponteiro para FILE

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

***Imprime** a string na saída padrão (stdout)*

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
       fputc (c, stdout);
   fclose (fd);
}
```

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

***Lê** caracteres da entrada padrão (stdin) e armazena na string passada como parâmetro*

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
      fputc (c, stdout);
   fclose (fd);
}
```

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

***open()** retorna um int como fd*

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
       fputc (c, stdout);
   fclose (fd);
}
```

***fopen()** retorna um ponteiro para FILE como fd*

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDO
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
```

Protótipo → ***int read (int fd, void *buf, int nbyte);***

- *fd → descritor de arquivo retornado por open()*
- ***&c*** *→ endereço da* ***char c***
- *1 → qtd de bytes a serem lidos (um char)*
- *Qdo o retorno de read() for igual a zero é pq todo o conteúdo do arquivo foi lido.*
- *Em C →* ***==*** *(igual) e* ***!=*** *(diferente)*

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
```

Protótipo → ***int fgetc (FILE * fd);***

- *fgetc retorna o caracter corrente (1 caracter) apontado pelo ponteiro de L/E do arquivo descrito por fd*
- *O ponteiro de arquivo é avançado automaticamente para o próximo caracter*
- *Se o fim do arquivo é alcançado, fgetc retorna EOF*

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char file
   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

A **atribuição em C/C++** produz um resultado, que é o mesmo valor atribuído ao alvo da atribuição. Portanto, uma instrução de atribuição pode ser usada como uma expressão e como operando em outras expressões

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   ts(filename);
   fd = pen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
```

Protótipo → ***int fgetc (FILE * fd);***

- *fgetc retorna o caracter corrente (1 caracter) apontado pelo ponteiro de L/E do arquivo descrito por fd*
- *O ponteiro de arquivo é avançado automaticamente para o próximo caracter*
- *Se o fim do arquivo é alcançado, fgetc retorna EOF*

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RD
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
```

Protótipo → ***int write (int fd, void *buf, int nbyte);***

- ***STDOUT*** → *constante = 1 que identifica a saída padrão; diferente de "stdout", que tb identifica a saída padrão, mas como um ponteiro FILE*
- ***&c*** → *endereço da* ***char c***
- ***1*** → *qtd de bytes a serem escritos (um char)*
- *write retorna o número de bytes escritos, mas isso não nos interessa aqui*

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c
   close(fd);
}
```

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
      fputc (c, stdout);
   fclose (fd);
```

Protótipo → ***int fputc (int c, FILE * fd);***

- *Escreve o caracter c em fd e avança o respectivo ponteiro de L/E*
- *Se algum erro ocorrer, retorna EOF*
- *Perceba que aqui foi usada a constante "stdout" correta*

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

close() fecha o arquivo cujo descritor é um inteiro (foi aberto com *open()*)

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
      fputc (c, stdout);
   fclose (fd);
}
```

Exemplo leitura/escrita em C

```
/* list.c - programa que le caracteres de
      um arquivo e escreve-os na tela */
#include <stdio.h>
#include <fcntl.h>
int main( )
{
   char c;
   int fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = open(filename, O_RDONLY);
   while(read(fd, &c, 1) != 0)
     write(STDOUT, &c, 1);
   close(fd);
}
```

```
/* versão "alto nível" */
#include <stdio.h>
int main( )
{
   char c;
   FILE* fd;
   char filename[20];

   printf ("De nome do arquivo: ");
   gets(filename);
   fd = fopen(filename, "r");
   while ((c=fgetc(fd)) != EOF)
       fputc (c, stdout);
   fclose (fd);
}
```

fclose() fecha o arquivo cujo descritor é um ponteiro FILE (foi aberto com *fopen()*)

Detecção de fim de arquivo

- Durante a leitura de um arquivo, o S.O. monitora a posição do ponteiro de L/E do arquivo
- Como detectar o fim de arquivo?
  - Em Pascal
    - Função booleana *eof()* que consulta o S.O. para saber se o ponteiro de L/E passou do último elemento do arquivo
    - Retorna *true* se o fim de arquivo foi encontrado e *false* caso contrário
  - Em C
    - Não tem uma função *eof()* nativa
    - Em vez disso, controla-se o fim de arquivo pelo retorno das funções de leitura (*read*(), *fgetc*(), ...)
      - As funções de leitura retornam 0 ou -1 quando atingem o fim do arquivo

Detecção de fim de arquivo

- Durante a leitura de um arquivo, o S.O. monitora a posição do ponteiro de L/E do arquivo
- Como detectar o fim de arquivo?
  - Em Pasca
    - Funçã
      pontei
    - Retorn
      contrá
  - Em C
    - Não te
    - Em vez
      funções de leitura (*read*(), *fgetc*(), ...)
      - As funções de leitura retornam 0 ou -1 quando atingem o fim do arquivo

`int feof ( FILE * stream )`

Para descritores FILE*, verifica o indicador de fim de arquivo

Um valor diferente de zero é retornado caso o indicador de fim de arquivo esteja setado. Caso contrário, o valor zero é retornado.

Esse indicador geralmente é setado por operações que tentam ler além do fim do arquivo.

Note que o ponteiro de L/E pode estar apontando o fim do arquivo, mas o indicador pode não estar setado ainda porque nenhuma operação tentou ler além desse ponto.

Exemplo de uso da feof()

```
#include <stdio.h>
int main () {
    FILE * pFile;
    int n = 0;
    pFile = fopen ("myfile.txt","rb");
    if (pFile==NULL) perror ("Error opening file");
    else {
        while (fgetc(pFile) != EOF) {
            ++n;
        }
        if (feof(pFile)) {
            puts ("End-of-File reached.");
            printf ("Total number of bytes read: %d\n", n);
        }
        else puts ("End-of-File was not reached.");
        fclose (pFile);
    }
    return 0;
}
```

Exemplo retirado de
http://www.cplusplus.com/reference/cstdio/feof/

Seeking

- Como mover o ponteiro de L/E?
  - Funções de leitura e escrita (read/write) movimentam automaticamente o ponteiro de L/E
  - Funções de posicionamento do ponteiro → ***seek***

- *Seeking* → ação de mover o ponteiro de L/E diretamente para uma determinada posição no arquivo
  - Requer pelo menos duas informações:
    - Nome do arquivo lógico no qual se quer fazer o *seek* → ***source_file***
    - Número de posições a partir do início do arquivo que se deseja mover o ponteiro de L/E → ***offset***

seek (source_file, offset)

Seeking em C

- Em C, um arquivo é visto como um vetor de bytes
  - Podemos posicionar o ponteiro de L/E em qualquer byte do arquivo

long lseek **(int** *fd*, **long int** *offset*, **int** *origin* **)**

    - Retorno: um inteiro igual a posição (em bytes) do ponteiro de L/E depois de ter sido movido
    - Três parâmetros:
      - **int** *fd*: descritor do arquivo *fd*
      - **long int** *offset*: número de bytes a partir da origem que se deseja mover o ponteiro de L/E
      - **int** *origin*: um valor que especifica a posição a partir da qual o *offset* deve ser calculado
        - » SEEK_SET → 0 → início do arquivo
        - » SEEK_CUR → 1 → posição corrente do ponteiro de L/E
        - » SEEK_END → 2 → fim do arquivo

Seeking em C

- Em C, um arquivo é visto como um vetor de bytes
  - Podemos posicionar o ponteiro de L/E em qualquer byte do arquivo

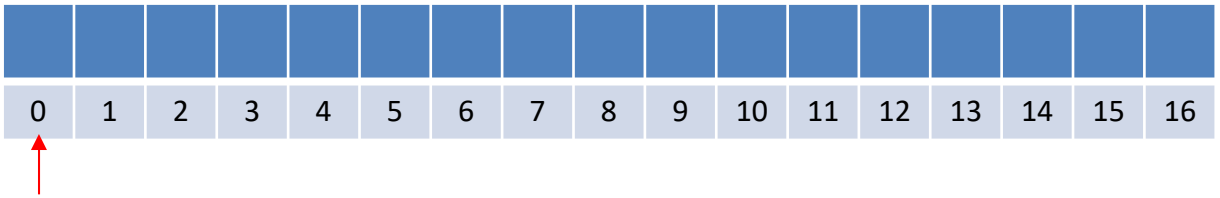
    long fseek **(FILE *** *fd*, **long int** *offset*, **int** *origin* **)**

    - Retorno: um inteiro igual a posição (em bytes) do ponteiro de L/E depois de ter sido movido
    - Três parâmetros:
      - **FILE*** *fd*: ponteiro para o arquivo *fd*
      - **long int** *offset*: número de bytes a partir da origem que se deseja mover o ponteiro de L/E
      - **int** *origin*: um valor que especifica de onde o deslocamento deve ser calculado
        - » SEEK_SET → 0 → início d
        - » SEEK_CUR → 1 → posiçã
        - » SEEK_END → 2 → fim do arquivo

No livro do Folk é utilizado o *lseek*, que é um comando do Unix incorporado na linguagem C.

fseek e *lseek* são equivalentes.

Exemplo *seeking* em C

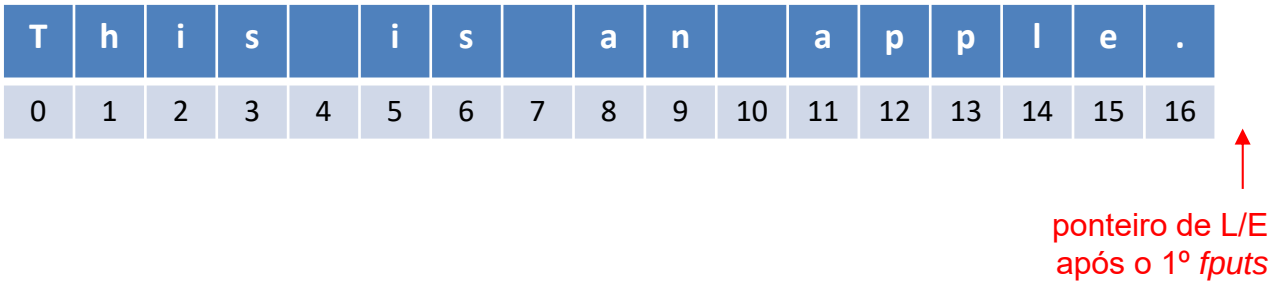



```
#include <stdio.h>

int main ()
{
  FILE * pFile;

  pFile = fopen ( "example.txt" , "w" );
  fputs ( "This is an apple." , pFile );
  fseek ( pFile , 9 , SEEK_SET );
  fputs ( " sam" , pFile );
  fclose ( pFile );
}
```

Exemplo *seeking* em C




```
#include <stdio.h>

int main ()
{
  FILE * pFile;

  pFile = fopen ( "example.txt" , "w" );
  fputs ( "This is an apple." , pFile );
  fseek ( pFile , 9 , SEEK_SET );
  fputs ( " sam" , pFile );
  fclose ( pFile );
}
```

Exemplo *seeking* em C

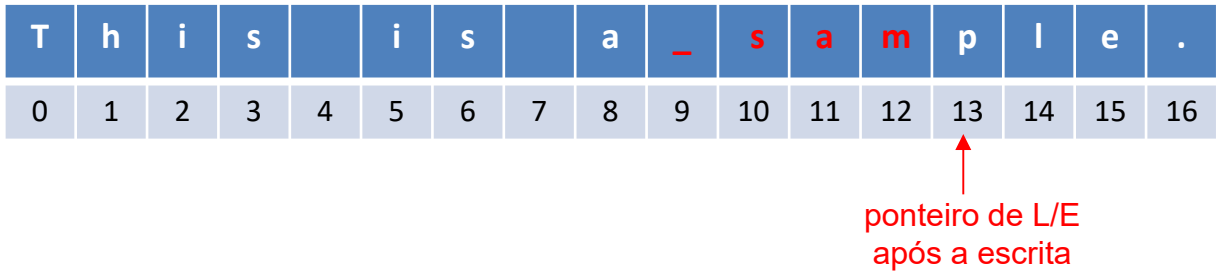

```
#include <stdio.h>

int main ()
{
  FILE * pFile;

  pFile = fopen ( "example.txt" , "w" );
  fputs ( "This is an apple." , pFile );
  fseek ( pFile , 9 , SEEK_SET );
  fputs ( " sam" , pFile );
  fclose ( pFile );
}
```

Nova posição do ponteiro de L/E → 0 + 9 = 9

Exemplo *seeking* em C




```
#include <stdio.h>

int main ()
{
  FILE * pFile;

  pFile = fopen ( "example.txt" , "w" );
  fputs ( "This is an apple." , pFile );
  fseek ( pFile , 9 , SEEK_SET );
  fputs ( " sam" , pFile );
  fclose ( pFile );
}
```

Escrita na nova posição, reposicionando o ponteiro de L/E

Seeking em C

- Outras funções de *seek* (stdio.h) para arquivos abertos com *"fopen"*
  - **long int** ftell **(FILE *** *fd* **)**
    - Retorna a posição atual do ponteiro de L/E como um inteiro
  - **void** rewind **(FILE *** *fd* **)**
    - Reseta o ponteiro de L/E de *fd* para o início do arquivo
    - Equivalente a um *fseek (fd , 0L , SEEK_SET)*

- Mais infos: http://www.cplusplus.com/reference/clibrary/cstdio/

Arquivos e o S.O.

- Marcação de fim de linha
  - Dependendo do S.O. utilizado, a marca de fim de linha pode mudar
    - DOS/Windows → par CR-LF (decimal ASCII 13 e 10 → hex 0D 0A)
    - Linux → LF (decimal ASCII 10 → hex 0A)
  - Para vermos essa marcação precisamos usar um editor hexadecimal
    - https://hexed.it/
    - https://www.onlinehexeditor.com/