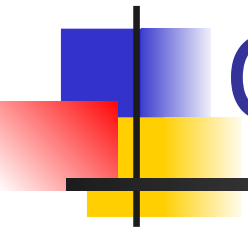


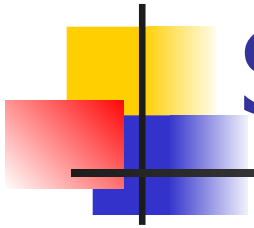
SCC0221 – Introdução à Ciência da Computação I

Prof.: Dr. Rudinei Goularte

(rudinei@icmc.usp.br)

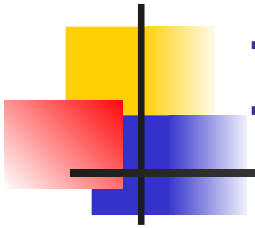
Arquivos

Instituto de Ciências Matemáticas e de Computação - ICMC
Sala 4-229



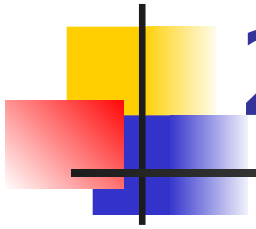
Sumário

- 1. Introdução.
- 2. Streams em C.
- 3. Arquivos em C.
- 4. Outros Comandos.



1. Introdução

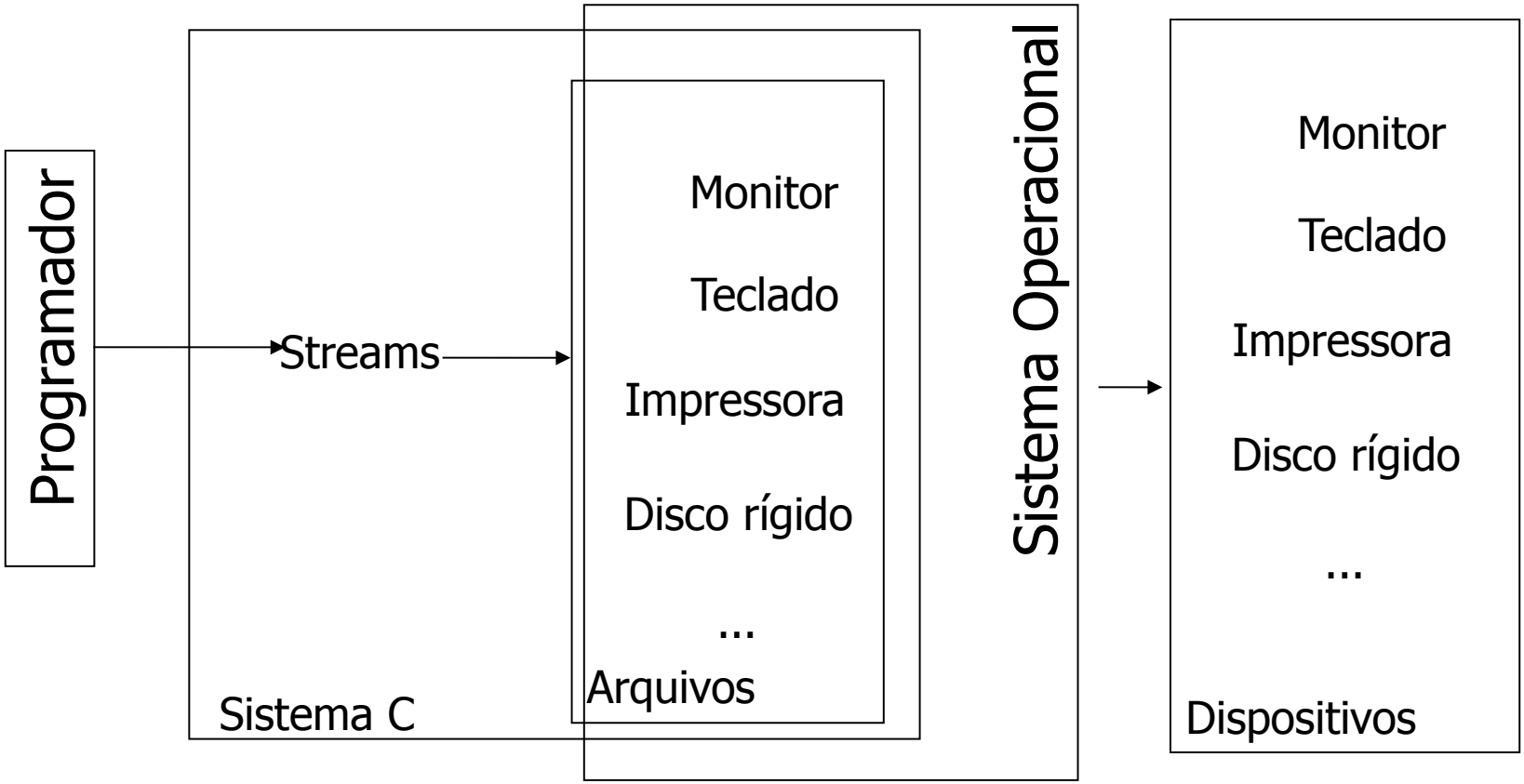
- O que é um arquivo?
  - Modelo para representar informações.
- Para que serve?
  - Armazenar informações de modo permanente em meio externo.
- Quando é necessário?
  - Armazenar informações de modo permanente.
  - Volume de dados muito grande.

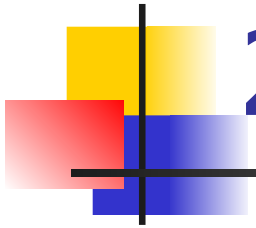


2. Streams em C

- Stream (significa fluxo ou seqüência)
  - Interface (abstração) entre programador e dispositivo fornecida pelo sistema de E/S em C.
  - Abstração = stream e dispositivo = arquivo.

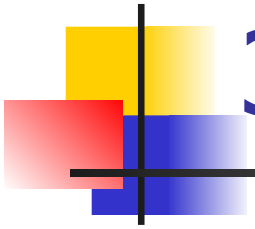
2. Streams em C





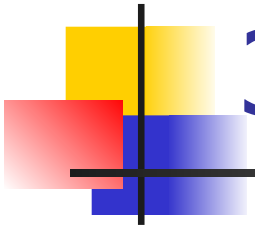
2. Streams em C

- Stream de texto
  - É uma seqüência de caracteres.
  - Caracter de nova linha & tabulações.
- Stream binária
  - É um seqüência de bytes.
  - Número de bytes escritos/lidos é o mesmo encontrado no dispositivo.



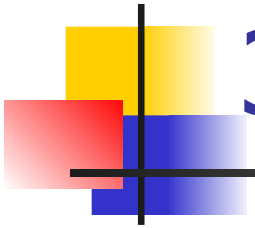
3. Arquivos em C

- Um arquivo em C pode ser qualquer coisa, desde um arquivo em disco (um .doc, p.e.) até uma impressora.
- Associação stream-arquivo é feita via operação de abertura.
- Após aberto, informações pode ser trocadas entre o programa e o arquivo.



3. Arquivos em C

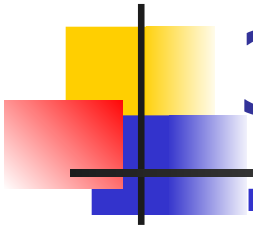
- Nem todos os arquivos apresentam os mesmos recursos.
  - Monitor x teclado.
- Arquivo é desassociado do stream via operação de fechamento.
  - Dados só serão escritos após fechamento.
  - Flushing, main() e exit(), crash e abort().



3. Arquivos em C

- Estrutura FILE
  - Estrutura de controle para streams.
  - Cabeçalho stdio.h.
  - Uso de FILE é feito via um ponteiro:

```
FILE *fp;
```



3. Arquivos em C

- Funções em stdio.h:
  - fopen()
  - fclose()
  - putc()
  - fputc()
  - getc()
  - fgetc()
  - fseek()
  - fprintf()
  - fscanf()
  - feof()
  - ferror()
  - rewind()
  - remove()
  - fflush()

3.1 Abertura de arquivo

- FILE *fopen(const char *nomearq,
  const char *modo);

Modo	**Significado**
r	Abre p/ leitura (texto)
w	Cria p/ escrita (texto)
a	Anexa ao fim (texto)
rb	Abre p/ leitura (binário)
wb	Cria p/ escrita (binário)
ab	Adiciona ao fim (binário)
r+	Abre p/ leitura/escrita (texto)
r+b	Abre p/ leitura/escrita (binário)

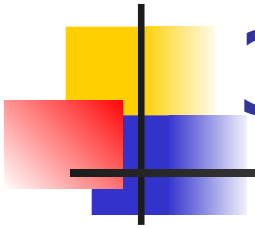
3.1 Abertura de arquivo

- Exemplo:

```
FILE *fp;
if ((fp = fopen("teste.dat", "w")) == NULL){
   printf("Erro na abertura do arquivo!");
   exit(1);
}
```

OBS.: diferença entre abrir p/ escrita e abrir para leitura e escrita, caso o arquivo já exista.

  - Com r+ ou r+b o arquivo existente não será apagado!



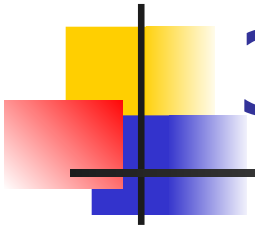
3.2 Fechamento de arquivo

- int fclose(FILE *fp);
- fclose() fecha um arquivo que foi aberto via fopen().
- Uma chamada à fclose():
  - Grava os dados (buffer).
  - Fecha o arquivo.
- Limite de arquivos abertos do SO.
  - Aconselhável fechar um antes de abrir outro.

3.2 Fechamento de arquivo

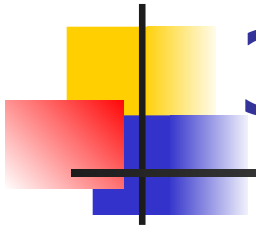
- Exemplo

```
FILE *fp;
if ((fp = fopen("teste.dat", "w")) == NULL){
   printf("Erro na abertura do arquivo!");
   exit(1);
}
...
fclose(fp);
```



3.3 Escrevendo em arquivos

- int putc (int ch, FILE *fp);
  - Sucesso: retorna o caracter (byte) lido.
  - Falha: devolve EOF.
- Idêntica a fputc()
- ch é int, mas apenas o bytes menos significativo é usado.

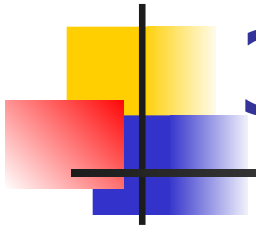


3.3 Escrevendo em arquivos

- Fazer um programa em C que escreva em um arquivo de texto utilizando a função fprintf(). Os caracteres a serem escritos devem ser obtidos do teclado. Digitar $ para terminar.

3.3 Escrevendo em arquivos

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
   FILE *fp;
   char c;

   if (argc != 2){
      printf("Faltou o nome do arquivo!");
      exit(1);
   }
   if((fp = fopen(argv[1],"w")) == NULL){
      printf("Erro na abertura do arquivo!");
      exit(1);
   }
   do{
      scanf("%c", &c);
      fprintf(fp, "%c", c);
   }while (c != '$');
   fclose(fp);
   return(0);
}
```

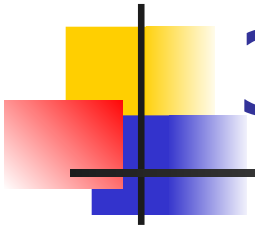


3.4 Lendo arquivos

- Fazer um programa em C que leia um arquivo de texto utilizando a função fscanf(). Os caracteres a serem escritos na tela devem ser obtidos do arquivo gravado no exemplo anterior.

3.4 Lendo arquivos

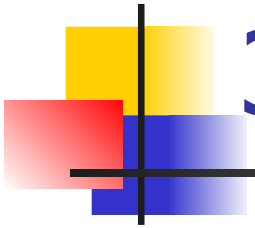
```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]){
   FILE *fp;

   char c;

   if (argc != 2){
      printf ("Uso: le nome_arq");
      exit(1);
   }

   if ((fp = fopen(argv[1], "r")) == NULL){
      printf("Erro na abertura do arquivo!");
      exit(1);
   }

   while (fscanf(fp, “%c”, &c) != EOF)
      printf(“%c”, c);

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



3.5 Fim de arquivo

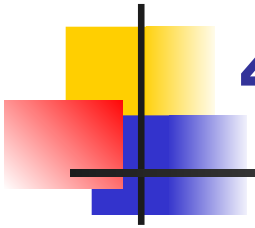
- int feof(FILE *fp);
  - Devolve verdadeiro se o fim de arquivo for encontrado.

```
while (!feof(fp))
   ch = getc(fp);
```



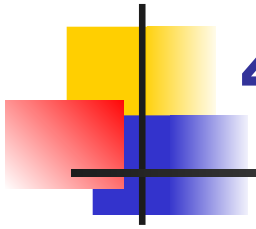
3.5 Fim de arquivo

- Fazer um programa em C (copia) que lê um arquivo copiando seu conteúdo para um outro arquivo. Abrir os arquivos para leitura em modo binário (arquivo 1) e para escrita em modo binário (arquivo 2). Utilizar a função feof().
- Dica, serão necessários 2 ponteiros para FILE, um para o arquivo sendo lido e outro para o arquivo sendo escrito.



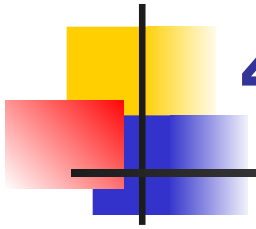
4. Outros comandos

- fputs() e fgets()
  - int fputs(const char *str, FILE *fp);
  - char *fgets(char *str, int length FILE *fp);
- rewind()
  - void rewind(FILE *fp);
- remove()
  - int remove(const char *filename);
  - Devolve 0 em caso de sucesso, e diferente de 0 caso contrário.



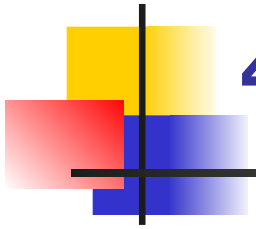
4. Outros comandos

- fflush()
  - int fflush(FILE *fp);
  - Esvazia um stream. Se for chamada com valor nulo, descarrega todos os streams abertos.



4. Outros comandos

- fread() e fwrite()
  - Permitem ler e escrever dados maiores que um byte.
  - size_t **fread**(void *buffer, size_t num_bytes, size_t count, FILE *fp);
  - size_t **fwrite**(const void *buffer, size_t num_bytes, size_t count, FILE *fp);
  - *size_t* está definido em stdio.h.



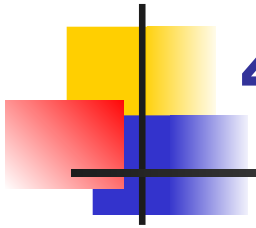
4. Outros comandos

- fread() e fwrite()
  - Podem ler ou escrever qualquer tipo de dado (int, long, double, struct, etc...).
  - Deve-se usar **sizeof**.

4. Outros comandos

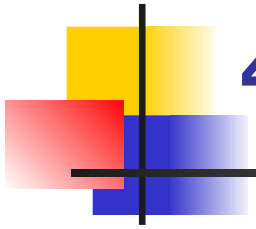
- Exemplos:

```
double num;
struct dados{
        char nome[50];
        int idade;
} dat;

fwrite(&num, sizeof(double), 1, fp);
fwrite(&dat, sizeof (struct dados),1, fp);

fread(&num, sizeof(double), 1, fp);
fread(&dat, sizeof(struct dados), 1, fp);
```



4. Outros comandos

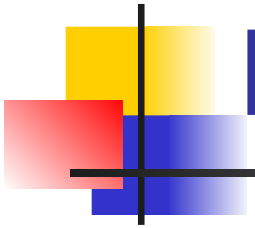
- fprintf() e fscanf()
  - Equivalentes a printf() e scanf().
  - int fprintf(FILE *fp, const char *control_str, ...);
  - int fscanf(FILE *fp, const char *control_str, ...);
- fseek()
  - Permite acesso aleatório.
  - int fseek(FILE *fp, long numbytes, int origin);
  - origin:
    - SEEK_SET
    - SEEK_CUR
    - SEEK_END
  - Devolve 0 se bem-sucedida, != de 0 caso contrário.



4. Outros comandos

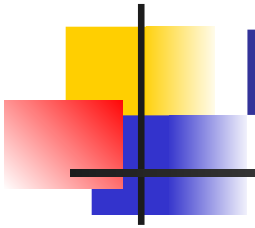
- Exemplo:
  - char c[] = “3”;

```
if (fseek(fp, atol(c), SEEK_SET)){
   printf(“Erro!”);
   exit(1);
}
```



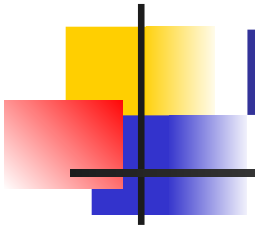
Exercício

- Fazer o programa Seek em C que, por linha de comando, receba a indicação do byte em que se deve posicionar um arquivo: seek nomearq byte
- O programa deve usar a função seek() para posicionar o arquivo no byte especificado, ler tal byte e imprimi-lo na forma de um caracter.



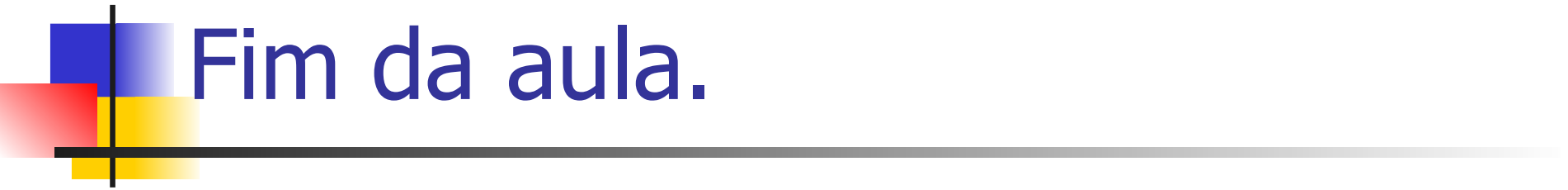
Exercícios para casa...

- 1) Escreva um programa para numerar as linhas de um arquivo texto. O nome do arquivo deve ser passado ao programa como um argumento de linha de comando. O programa deve escrever para *stdout* (a saída padrão - o monitor). Cada linha do arquivo de entrada deve ser direcionada para o arquivo de saída (stdout) com o número da linha e um espaço como prefixos.



Exercícios para casa...

- 2) Escreva um programa que mostre um arquivo na tela 20 linhas por vez. O nome do arquivo deve ser passado ao programa como um argumento de linha de comando. O programa deve mostrar as próximas 20 linhas após a tecla <return> ter sido pressionada. (Essa é uma versão rudimentar do utilitário *more* do UNIX).



Fim da aula.