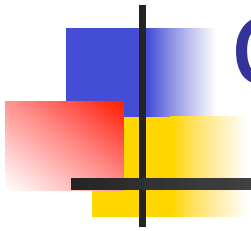
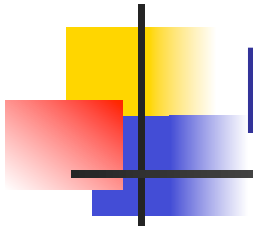


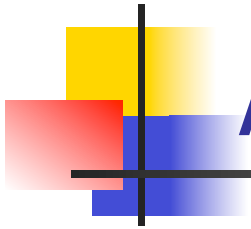
Operações de Processamento de Arquivos

Thiago A. S. Pardo
Leandro C. Cintra
M.C.F. de Oliveira
Moacir Ponti Jr.
Cristina D. A. Ciferri



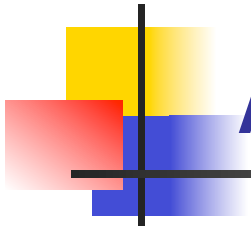
Fundamentos de Arquivos

- Posição corrente no arquivo
- Descritor de arquivo
- *File handle*



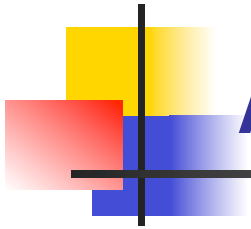
Arquivo Físico e Arquivo Lógico

- **Arquivo Físico**
  - sequência de bytes armazenados no disco, geralmente agrupados em setores
  - gerenciado pelo sistema operacional
- **Arquivo Lógico**
  - arquivo como visto pelo aplicativo que o acessa (como a linguagem de programação vê os dados)
- **Associação**
  - iniciada pelo aplicativo
  - gerenciada pelo sistema operacional



Arquivos em C: bibliotecas

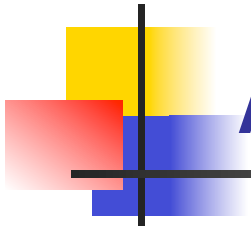
- 2 níveis de manipulação
  - Funções de baixo nível
  - Funções de alto nível (stdio.h)
    - Implementadas em baixo nível
    - Bufferizadas: mantêm área de memória para manipulação dos bytes



Arquivos em C

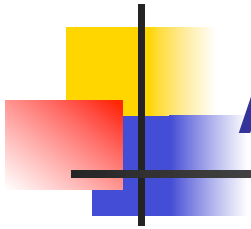
- Dev-C++

```
#include <stdio.h>

#define FOPEN_MAX (20)

typedef struct _iobuf
{
    char*   _ptr;
    int     _cnt;
    char*   _base;
    int     _flag;
    int     _file;
    int     _charbuf;
    int     _bufsiz;
    char*   _tmpfname;
} FILE;
```



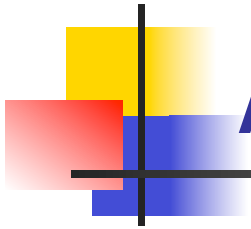
Arquivos em C

- Abertura
  - FILE *fd;
  - fd =fopen(<filename>,<flags>);
  - flags: controla o modo de abertura
    - "r“ Abre apenas para leitura; o arquivo precisa existir
    - "w“ cria arquivo vazio para escrita (se já existe, é apagado)
    - "a“ adiciona conteúdo ao arquivo (append)
    - "r+“ Abre arquivo para leitura e escrita
    - "w+“ Cria arquivo vazio para leitura e escrita
    - "a+“ Abre arquivo para leitura e adição (append)
      - Se b depois de r/w/a, modo binário
      - Caso contrário (sem b), modo texto (t)



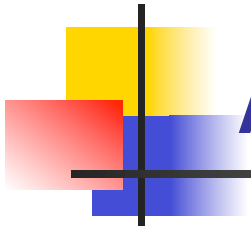
Arquivos em C

- Modo texto vs. binário
  - Em texto, \n e \r são gravados como dois bytes, mas convertidos na leitura para apenas um \n; em modo binário, não há essa conversão
  - Em texto, fim de arquivo é o Ctrl-Z; em modo binário, é simplesmente mais um caractere, o fim pode ser qualquer caractere
  - Em binário, número é gravado e lido exatamente como está na memória; em texto, como ASCII
  - Algumas funções de arquivo são destinadas a um ou outro tipo



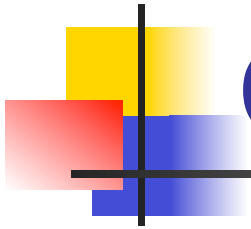
Arquivos em C

- Fechamento
  - fclose(fd)
    - Transfere o restante da informação no buffer e fecha o arquivo
      - Por que se precisa de um buffer, afinal?



Arquivos em C

- Grupos de funções para leitura de arquivos
  1. Caractere a caractere
  2. Linha a linha
  3. Dados formatados
  4. Blocos de bytes



Caractere a caractere

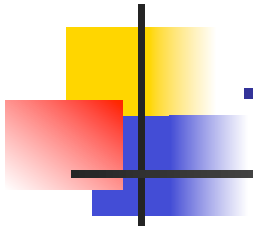
- fputc(<char>,<FILE>)
  - escreve um caractere no arquivo
- <char> = fgetc(<FILE>)
  - lê um caractere do arquivo
- EOF: caractere de fim de arquivo
  - feof(<FILE>)
    - função que retorna 1 se fim de arquivo

Caractere a caractere - fputc e fgetc

```
int main(void) {
        FILE *fp;
        char ch;
        fp=fopen(“teste.txt”,“wt”);
        while ((ch=getche())!= ‘\r’ )
            fputc(ch,fp)
        fclose(fp);
        return 0;
}
```

```
int main(void) {
        FILE *fp;
        char ch;
        fp=fopen(“teste.txt”,“rt”);
        while ((ch=fgetc(fp))!=EOF)
            printf(“%c”,ch);
        fclose(fp);
        system(“pause”);
        return 0;
}
```

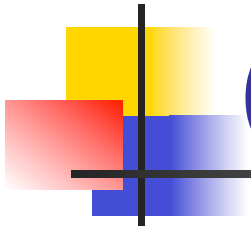
Caractere a caractere - feof



- EOF indica fim de arquivo
- A função feof verifica se se atingiu o final do arquivo

```
...
while (!feof(fp)) {
    ch=fgetc(fp);
    printf("%c",ch);
}
...
```

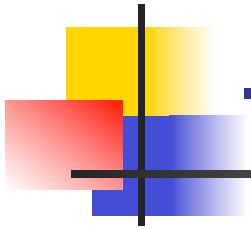
Linha a linha (cadeia caracteres)



- fputs(<char *>,<FILE>)
  - escreve uma string no arquivo
- fgets(<char *>,<int>,<FILE>)
  - lê do arquivo uma determinada quantidade de caracteres e armazena a cadeia de caracteres em uma variável
  - retorna NULL se fim de arquivo

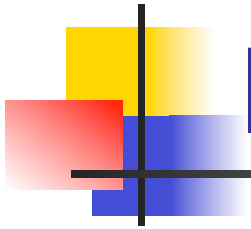
Atenção: fputs não adiciona o \n automaticamente

Linha a linha - fputs e fgets



```
int main(void) {
        FILE *fp;
        fp=fopen(“teste.txt”,“wt”);
        fputs(“Isso é um teste\n”,fp);
        fputs(“Vamos testar!\n”,fp);
        fclose(fp);
        system(“type teste.txt”);
        system(“pause”);
        return 0;
}
```

```
int main(void) {
        FILE *fp;
        char str[100];
        fp=fopen(“teste.txt”,“rt”);
        while (fgets(str,100,fp)!=NULL)
              printf(“%s”,str)
        fclose(fp);
        system(“pause”);
        return 0;
}
```



Dados formatados

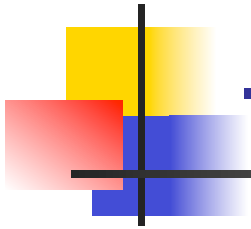
- fprintf(<FILE>,formatacao, ...)
  - escreve em um arquivo texto a string formatada
  - *similar ao printf()*
- fscanf(<FILE>,formatacao, ...)
  - lê do arquivo strings formatadas
  - retorna EOF se fim de arquivo
  - *similar ao scanf()*

Dados formatados - fprintf e fscanf

- Catálogo de livros: título, número de registro, preço

```
int main(void) {
        FILE *fp;
        char titulo[30];
        int reg;
        float preco;
        fp=fopen("teste.txt","wt");
        ...
        ...
        while (1) {
            prinf("Digite título, registro e preço: ");
            scanf("%s %d %f",titulo,&reg,&preco);
            if (strlen(titulo)<=1) break;
            fprintf(fp,"%s %d %f\n",titulo,reg,preco);
        }
        fclose(fp);
        return 0;
}
```

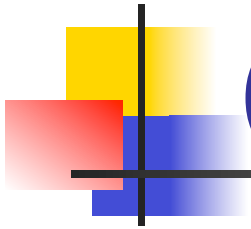
Dados formatados - fprintf e fscanf



- Catálogo de livros: título, número de registro, preço

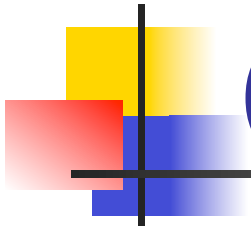
```
int main(void) {
        FILE *fp;
        char titulo[30];
        int reg;
        float preco;
        fp=fopen("teste.txt","rt");
        while (fscanf(fp,"%s %d %f\n",titulo,&reg,&preco)!=EOF)
              printf("%s %d %f\n",titulo,reg,preco);
        fclose(fp);
        system("pause");
        return 0;
}
```

Blocos de bytes (arquivos binários)

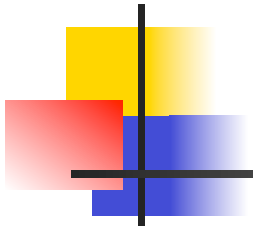


- <size_read> = fread(<buffer>,<size_un>, <size_buffer>, <FILE>)
  - size_read: unidades lidas (0 se fim de arquivo)
  - buffer: variável que vai armazenar a leitura
  - size_un: tamanho de cada unidade (bloco) de bytes a ser lido
  - size_buffer: número de blocos a serem lidos
  - FILE: ponteiro FILE

Blocos de bytes (arquivos binários)

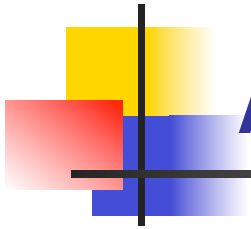


- <size_write> = fwrite(<buffer>,<size_un>, <size_buffer>, <FILE>)
  - size_write: unidades escritas (0 se não tiver espaço em disco)
  - buffer: variável que vai armazenar a escrita
  - size_un: tamanho de cada unidade (bloco) de bytes a ser escrito
  - size_buffer: número de blocos a serem escritos
  - FILE: ponteiro FILE



Blocos de bytes - fwrite e fread

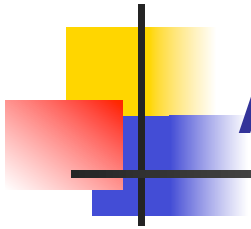
```
int main(void) {
        FILE *origem, *destino;
        char dados[1024];
        int bytes_lidos;
        origem=fopen("teste1.txt","rb");
        destino=fopen("teste2.txt","wb");
        do {
             bytes_lidos=fread(dados, sizeof(char), sizeof(dados), origem);
             fwrite(dados, sizeof(char), bytes_lidos, destino);
        } while (bytes_lidos);
        fclose(origem);
        fclose(destino);
        return 0;
}
```



Arquivos em C

- Lendo e gravando estruturas em arquivos

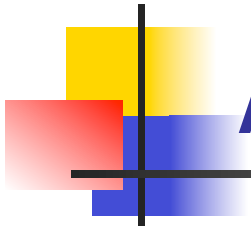
```
typedef struct livro {
    char titulo[50];
    int reg;
    float preco;
} Livro;
...
int main(void) {
    Livro livro;
    FILE *fp;
    fp=fopen("teste.txt","wb");
    livro=GetLivro();  //função GetLivro definida em outro lugar
    fwrite(&livro,sizeof(Livro),1,fp);
    fclose(fp);
    return 0;
}
```



Arquivos em C

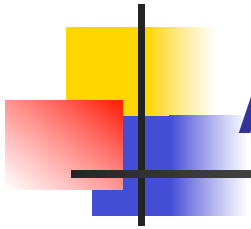
- Lendo e gravando estruturas em arquivos

```
typedef struct livro {
    char titulo[50];
    int reg;
    float preco;
} Livro;
...
int main(void) {
    Livro livro;
    FILE *fp;
    fp=fopen("teste.txt","rb");
    while (fread(&livro,sizeof(Livro),1,fp))
        printf("%s %d %f\n",livro.titulo,livro.reg,livro.preco);
    fclose(fp);
    system("pause");
    return 0;
}
```



Arquivos em C

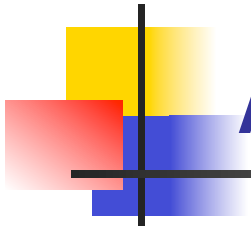
- Leitura sequencial
  - ponteiro de leitura avança byte a byte (ou por blocos), a partir de uma posição inicial
- Acesso aleatório (direto)
  - acesso envolve o posicionamento do ponteiro em um byte ou registro arbitrário



Arquivos em C

- Outras funções importantes
  - rewind
    - Posiciona o ponteiro do arquivo no início dele
      - Boa para quando se quer ler e escrever ao mesmo tempo no arquivo

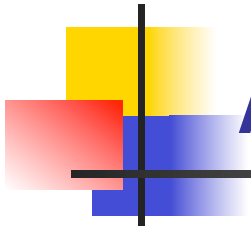
```
FILE *fp;
...
rewind(fp);
...
```



Arquivos em C

- Outras funções importantes
  - fflush
    - Esvazia o buffer, transferindo seu conteúdo para o arquivo
      - Recomendado voltar o ponteiro de arquivo para o início

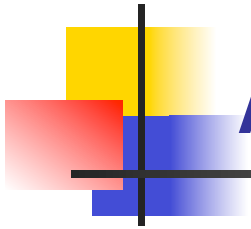
```
...
FILE *fp;
...
fflush(fp);
...
```



Arquivos em C

- Outras funções importantes
  - fseek
    - Movimenta o ponteiro de arquivo para a posição escolhida
      - fseek(arquivo, deslocamento em bytes, posição a partir de onde desloca)
        - posição 0 = SEEK_SET (início do arquivo)
        - posição 1 = SEEK_CUR (posição atual)
        - posição 2 = SEEK_END (fim do arquivo)

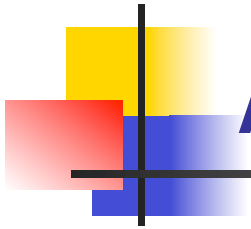
```
...
FILE *fp;
...
fseek(fp,0,SEEK_SET);   //volta ao byte 0 (início) do arquivo
...
```



Arquivos em C

- Outras funções importantes
  - ftell
    - Retorna a posição corrente do ponteiro do arquivo (em número de bytes a partir do início)
      - Só funciona bem com arquivos em modo binário

```
...
FILE *fp;
long pos;
...
pos=ftell(fp);
...
```



Arquivos em C

- Atenção
  - É importante fazer verificações de erros sobre os valores retornados pelas funções
    - Por exemplo, fopen retorna NULL se não conseguir abrir um arquivo

```
...
FILE *fp;
if ((fp=fopen("teste.txt","r"))==NULL)
        exit(1);
...
```

  - Quando é que não é possível abrir um arquivo?