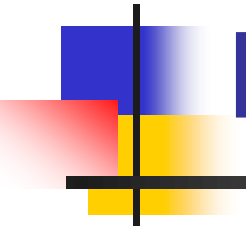
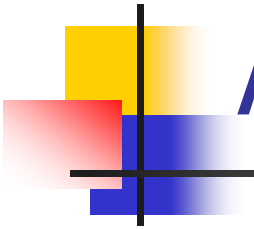


Fundamentos de Arquivos

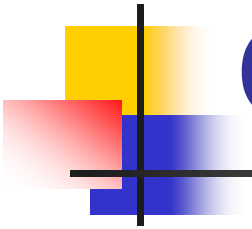
Estruturas de Dados II

Slides cedidos pelo Prof. Gustavo Batista (ICMC-USP)
[http://www.icmc.usp.br/~gbatista/]



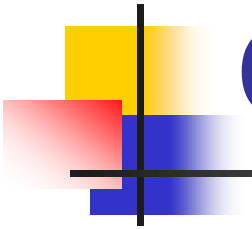
Arquivos

- Mecanismo de organização da informação mantida em memória secundária:
  - HD;
  - Fitas;
  - CD;
  - DVD.
- Da mesma forma que variáveis e alocação dinâmica organizam o acesso do programa à memória principal.



Organização de Arquivos

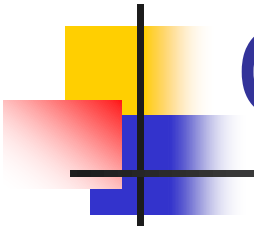
- Explorar a contrapartida entre as vantagens e desvantagens do disco é o ponto chave no projeto de estruturas de dados em arquivos.
- Meta: criar estruturas para minimizar as desvantagens do uso da memória externa (i.e., minimizar o tempo de acesso).



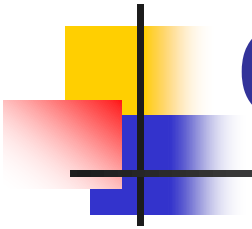
Organização de Arquivos

- Idealmente se deseja encontrar a informação em um único acesso.
- Se não for possível, deve-se encontrar em poucos acessos.
  - Mesmo uma busca binária não é eficiente o suficiente (~16 acessos para 50.000 registros).
- Deseja-se preferencialmente encontrar todas as informações necessárias em uma única leitura.

Organização de Arquivos



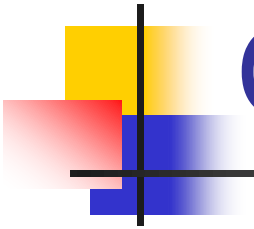
- Estruturas de dados eficientes em memória principal são inviáveis em memória secundária.
- Seria fácil obter uma estrutura de dados adequada para disco se os arquivos fossem estáticos (não sofressem alterações).



Organização de Arquivos

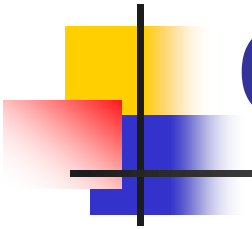
- Por exemplo, árvores AVL são consideradas rápidas para pesquisa em memória principal, mas mesmo com árvores balanceadas, dezenas de acessos ao disco são necessários.

Organização de Arquivos



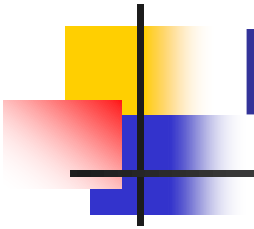
- Somente quase 10 anos mais tarde foram criadas as árvores-B. Um dos motivos da demora foi que a abordagem para criar árvores-B é bastante diferente das outras árvores (árvores-B tem crescimento bottom-up).
  - Árvores-B são a base para implementações comerciais de diversos sistemas (incluindo SGBDs).
  - Em termos práticos, permitem encontrar uma entrada entre milhões em 3 ou 4 acessos.

Organização de Arquivos



- Outras técnicas como hashing externo permitem acessos mais rápidos para arquivos que não sofrem muitas alterações.
- Contudo, hashing dinâmico tem sido aplicado para acesso rápido mesmo em grandes arquivos.

Arquivo Físico e Arquivo Lógico



- Arquivo Físico: sequência de bytes armazenada no disco em blocos distribuídos em trilhas/setores.
- Arquivo Lógico: arquivo como visto pelo aplicativo que o acessa – sequência contínua de registros ou bytes.
  - Cada arquivo manipulado pelo aplicativo é tratado como se um canal de comunicação (uma linha telefônica?) fosse estabelecido entre o aplicativo e o arquivo em disco
- Associação arquivo físico – arquivo lógico: iniciada pelo aplicativo, gerenciada pelo S.O.

Arquivo Físico e Arquivo Lógico

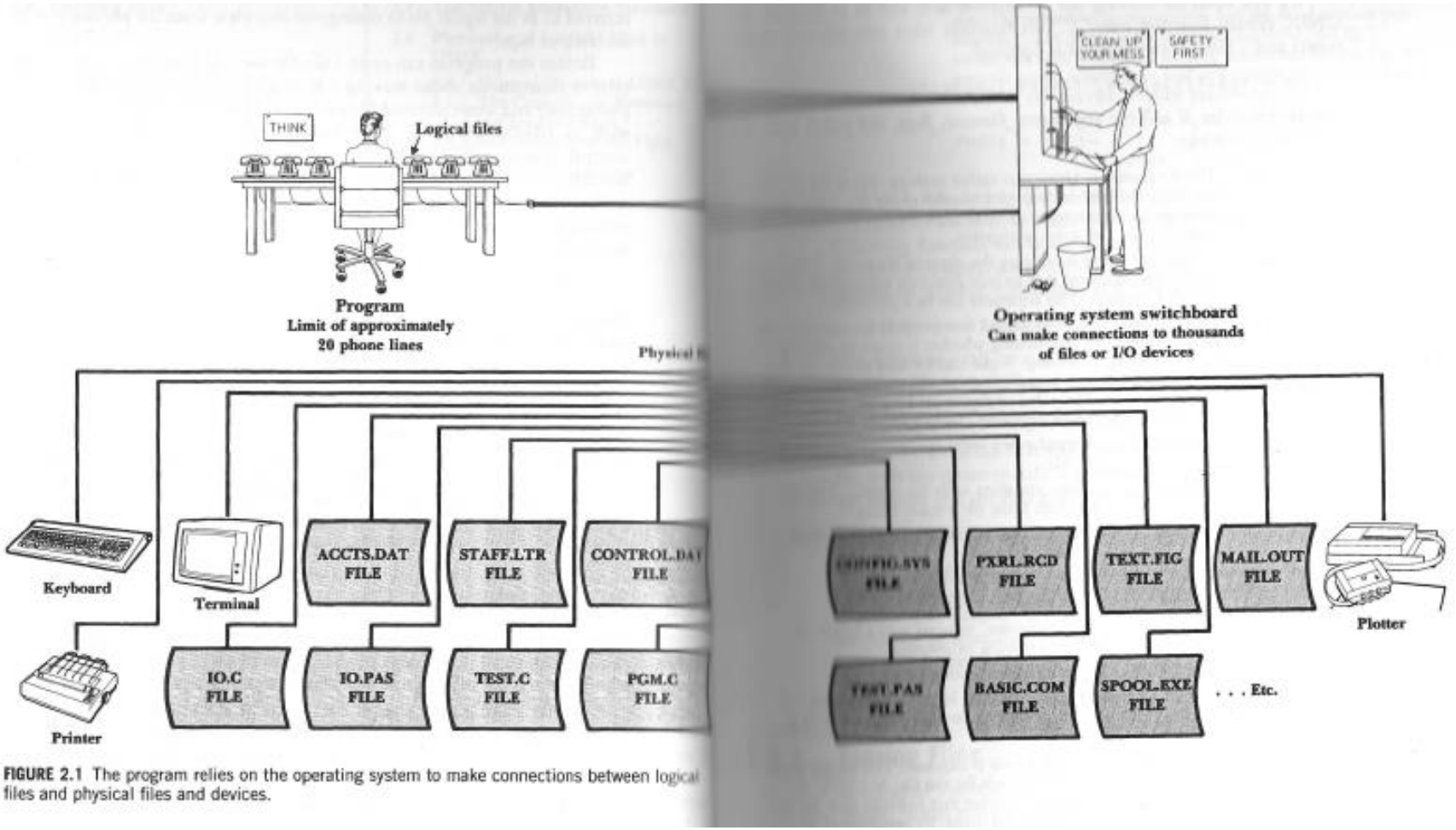
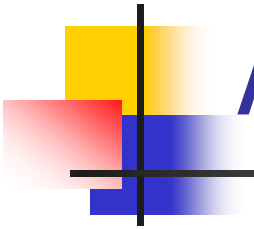


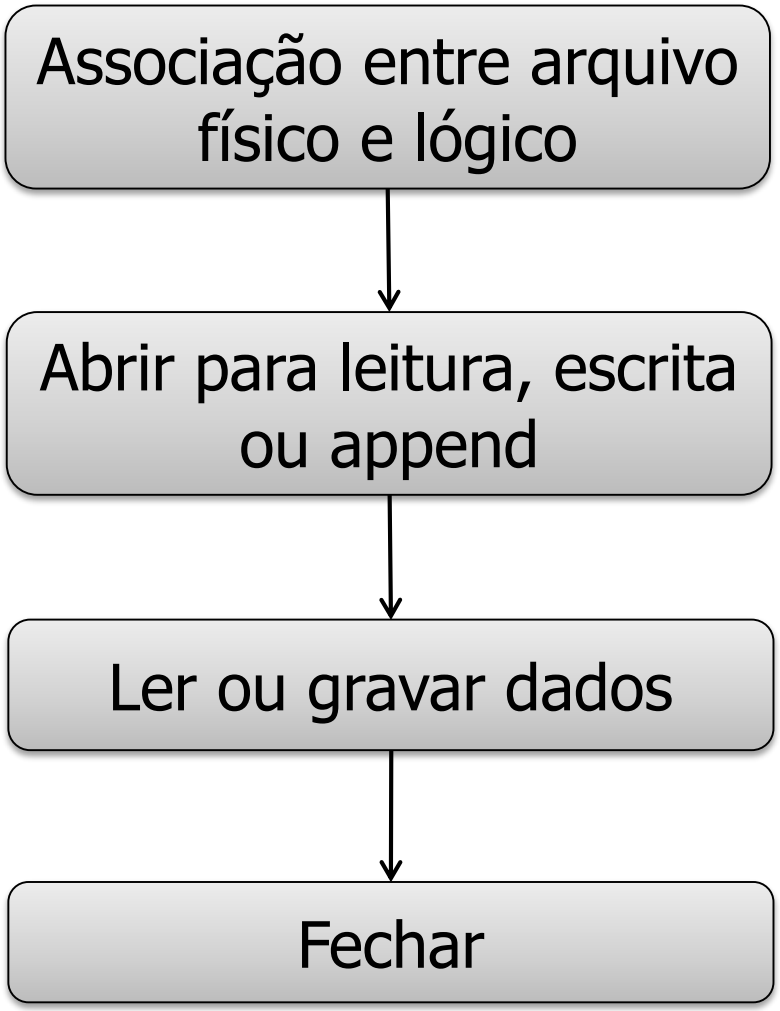
FIGURE 2.1 The program relies on the operating system to make connections between logical files and physical files and devices.



Arquivo Lógico

- A maioria das linguagens de programação tratam as informações que são lidas e gravadas em arquivos como streams de bytes.
- Gerenciar arquivos é portanto gerenciar esses streams.
- Algumas operações comuns sobre arquivos são abrir, fechar, ler e gravar além de verificar situações de erro.

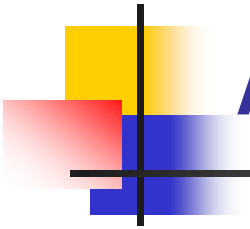
Operações sobre Arquivos



Associação entre Arquivo Físico e Arquivo Lógico

- Em C: (associa e abre para leitura)

```
FILE *p_arq;
if ((p_arq=fopen("meuarq.dat", "r"))==NULL)
   printf("Erro na abertura do arquivo\n");
else
```



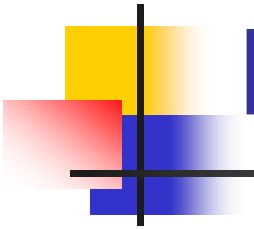
Abertura de Arquivos

- Em C
  - Comandos
    - fopen – comando da linguagem (<stdio.h>)
  - Parâmetros indicam o modo de abertura

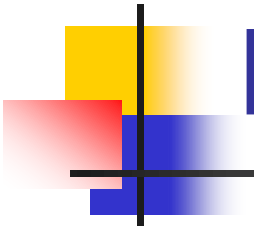
Função fopen

- fd=fopen(<filename>,<flags>)
  - filename: nome do arquivo a ser aberto;
  - flags: controla o modo de abertura;
    - "r": Abre para leitura. O arquivo precisa existir;
    - "w": Cria um arquivo vazio para escrita;
    - "a": Adiciona conteúdo ao arquivo (append);
    - "r+": Abre o arquivo para leitura e escrita;
    - "w+": Cria um arquivo vazio para leitura e escrita;
    - "a+": Abre um arquivo para leitura e adição (append);
    - "t": modo texto (default) – o fim do arquivo é o primeiro CTRL+Z (DOS) encontrado;
    - "b": modo binário – o fim do arquivo é o último byte.
- Caso o arquivo possa ser aberto, a função retorna um ponteiro para a estrutura FILE contendo as informações sobre o arquivo. Caso contrário, retorna NULL.

Função fopen



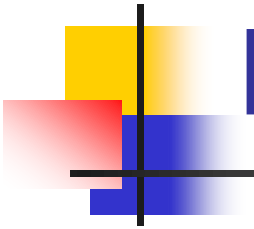
- Diferenças
  - Modo texto
    - Sequência de caracteres agrupadas em linhas
    - Linhas separadas por um único caractere
      - DOS/Windows: \r\n (CR (carriage return) (13) + LF (line feed) (10))
      - Unix: \n
      - Mac: \r
  - Modo binário
    - Os dados são armazenados em arquivos binários da mesma forma que são armazenados na memória do computador. Ou seja, ao guardarmos o valor de uma variável float num arquivo binário, ela ocupa no arquivo os mesmos 4 bytes que ocupa na memória



Função fopen

- Diferenças (cont.)
  - Modo texto
    - Compilador converte o valor da variável para uma forma de apresentá-lo através de caracteres
    - Por exemplo, o número inteiro 20000, apesar de ocupar somente dois bytes na memória (0000.0111.1101.0000), para ser armazenado em um arquivo texto ocupa pelo menos cinco bytes, já que cada caractere ocupa um byte
  - Arquivos binários normalmente são menores que arquivos texto contendo a mesma informação; assim, a velocidade de leitura do arquivo também é consideravelmente reduzida ao utilizarmos arquivos binários
  - Em compensação, a visualização do arquivo não traz nenhuma informação ao usuário, pelo fato de serem visualizados os caracteres correspondentes aos bytes armazenados na memória. No caso de variáveis char, a visualização é idêntica em arquivos texto e binários.

Fechamento de Arquivos



- Encerra a associação entre arquivos lógico e físico, garantindo que todas as informações sejam atualizadas e salvas (conteúdo dos buffers de E/S enviados para o arquivo).

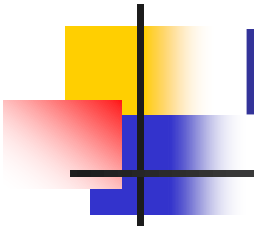
- S.O. fecha o arquivo se o aplicativo não o fizer ao final da execução do programa. Interessante para:
  - Garantir que os buffers sejam descarregados (prevenção contra interrupção);
  - Liberar as estruturas associadas ao arquivo para outros arquivos.

Exemplo: fechamento de arquivos

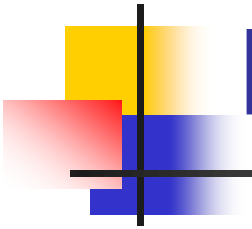
- C: fclose
  - Em caso de êxito, a função retorna 0. Caso contrário, retorna EOF (*end-of-file*) (=-1).

```
fd= fopen("meuarq.dat", "r")
...
fclose(fd)
```

Leitura e Escrita



- C: Funções da linguagem
  - Básicas (registro)
    - fread(&<dest-addr>, sizeof(<dest-addr>), <nr>, FILE* f)
      - A função retorna o número de unidades efetivamente lidas. Este número pode ser menor que <nr> quando o fim do arquivo for encontrado ou ocorrer algum erro (== 0).
    - fwrite(&<source-addr>, sizeof(<source-addr>), <nr>, FILE* f)
      - A função retorna o número de itens escritos. Este valor será igual a <nr> a menos que ocorra algum erro (== 0).



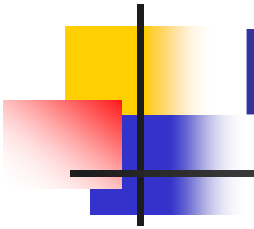
Leitura e Escrita

- C: Funções da linguagem
  - Caracteres
    - int fgetc(FILE* f);
      - Retorna o caractere lido. Em caso de erro, retorna EOF
    - int fputc(int ch, FILE* f);
      - O valor retornado é o próprio caractere fornecido. Em caso de erro, a função retorna EOF
  - Formatadas
    - **int** fscanf(FILE* f, <formatação>, variáveis)
      - Em caso de erro, retorna EOF
    - **int** fprintf(FILE* f, <formatação>, variáveis)
      - Em caso de erro, retorna EOF

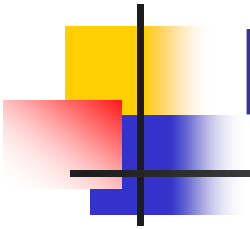
Leitura e Escrita

- C: Funções da linguagem
  - String
    - **char*** fgets(**char*** s, **int** n, FILE* f);
      - A função pára de ler quando lê *n-1* caracteres ou o caractere LF ('\n'), o que vier primeiro. O caractere LF é incluído na string. O caractere nulo é anexado ao final da string para marcar seu final. Em caso de êxito, a função retorna *s*. Em caso de erro ou fim do arquivo, retorna NULL.
    - **int** fputs(**char*** s, FILE* f);
      - O valor retornado é o último caractere fornecido. Em caso de erro, a função retorna EOF. Note que a função copia a string fornecida tal como ela é para o arquivo. Se não existir na string, não é inserido o caractere de nova linha (LF) nem o caractere nulo.

Fim de Arquivo



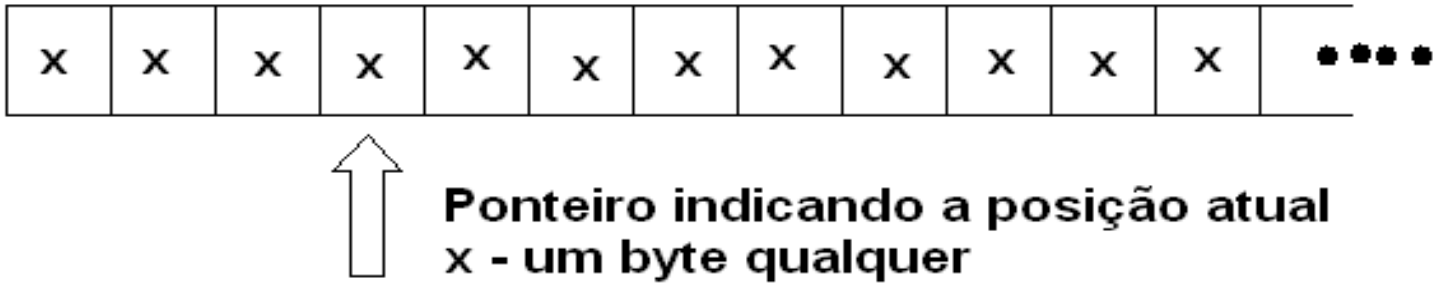
- Ponteiro de arquivo: controla o próximo byte a ser lido.
  - C:
    - feof(arq) (função lógica)
      - Retorna um valor inteiro caso o ponteiro chegou no final do arquivo, caso contrário o retorno é zero.
      - Problema escrita duplicada
    - fread() retorna o número de bytes lidos. Se igual a zero, indica final de arquivo.



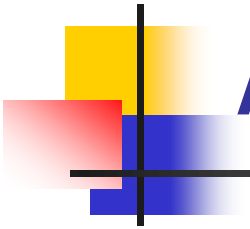
Exemplos

- Programas leitura/escrita
  - fgetc/fputc
  - fgets/fputs
  - fprintf/fscanf
  - fread/fwrite

O ponteiro no arquivo lógico



Acesso sequencial X aleatório



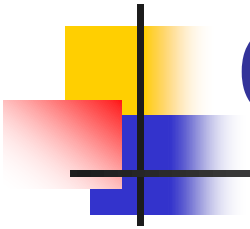
- Leitura sequencial: ponteiro de leitura avança byte a byte (ou por blocos), a partir de uma posição inicial

- Acesso aleatório (direto): posicionamento em um byte ou registro arbitrário

Seeking

- Seeking: ação de mover o ponteiro para uma certa posição no arquivo:
  - C:
  - **int** fseek(FILE* f,<byte-offset>,<origem>)
    - Função retorna a posição final do ponteiro;
      - Em caso de sucesso, retorna 0; caso contrário, retorna -1
    - byte-offset – deslocamento, em bytes, a partir de origem;
    - Origem: 0 – início do arquivo; 1 – posição corrente; 2 – final do arquivo.

Outras funções



- C:
  - **long** ftell(FILE* f)
    - Retorna um valor do tipo long, que representa o número de bytes do começo do arquivo até a posição atual
  - **void** rewind(FILE* f)
    - Retorna o ponteiro do arquivo para o início do arquivo