

# Fundamentos de Arquivos

---

SCC203 – Algoritmos e Estruturas de Dados II



# Arquivos

---

- Informação mantida em memória secundária
  - HD
  - Disquetes
  - Fitas magnéticas
  - CD
  - DVD



# Discos X Memória Principal

---

- Tempo de acesso
  - **HD**: ~ microsegundos  $\mu\text{s}$  ( $10^{-6}$ )
  - **RAM**: ~ nanosegundos  $\text{ns}$  ( $10^{-9}$ )
  - HDs são centenas – e até milhares – de vezes mais lentos que memória RAM



# Discos X Memória Principal

---

- Exemplo:
  - O acesso à RAM equivale a buscar uma informação no índice de um livro que está em suas mãos
  - O acesso a disco seria equivalente a mandar buscar a mesma informação em uma biblioteca



# Discos X Memória Principal

---

- Capacidade de Armazenamento
  - HD – muito alta, a um custo relativamente baixo
  - RAM – limitada pelo custo e espaço
- Tipo de Armazenamento
  - HD – não volátil
  - RAM – volátil



# Discos X Memória Principal

---

- Em resumo
  - **acesso a disco é muito caro**, isto é, lento!
- Então
  - o número de acessos ao disco deve ser **minimizado**
  - a quantidade de informações recuperadas em um acesso deve ser **maximizada**
- Estruturas de organização de informação em arquivos!



# Organização de Arquivos

---

- Meta: minimizar as desvantagens do uso da memória externa
  - Minimizar o tempo de acesso ao dispositivo de armazenamento externo

- De forma independente da tecnologia

Tempo de Acesso = nro. de acessos \* tempo de 1 acesso



# Discos X Memória Principal

---

- Estruturas de dados eficientes em memória principal são **inviáveis** em disco
- Seria fácil obter uma estrutura de dados adequada para disco se os arquivos fossem estáveis (não sofressem alterações)
  - Solução: **organização adequada de arquivos no disco, e de informações em arquivos**



# Discos X Memória Principal

---

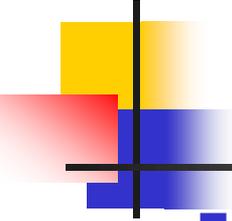
- O ideal é que a informação necessária possa ser obtida com **apenas 1 acesso** a disco.
  - Se o ideal não pode ser atingido, deseja-se chegar o mais próximo possível.
- Por exemplo, o método de busca binária permite que um registro pesquisado entre 50.000 seja encontrado em no máximo 16 comparações ( $\log_2 50.000 \sim 16$ ) .... mas acessar o disco 16 vezes para buscar uma informação é tempo demais. Precisamos de estruturas que permitam recuperar esse mesmo registro em **dois ou três acessos!**



# Discos X Memória Principal

---

- Queremos estruturas que **agrupem** informações de modo a permitir que toda (ou quase toda) a informação necessária seja obtida, idealmente, em uma única operação de acesso a disco
- Por exemplo, se precisamos do nome, endereço, telefone, saldo, número da conta, etc. de um certo cliente, é preferível obter toda essa informação de uma só vez ao invés de ficar procurando em vários lugares...



# Discos X Memória Principal

---

## ■ História - Cronologia

- Dados em **fitas** com acesso sequencial
- Arquivos cresceram demais e o **acesso sequencial ficou proibitivo**
- Uso de **índices** que, com o crescimento dos arquivos, também ficam ineficientes
- Uso de **árvores** para apontar para os arquivos, mas árvores crescem de maneira desigual
- **AVLs**
  - Qual a dificuldade?
- **Árvores B** e árvores B+
- **Hashing** seria uma boa opção, mas arquivos não são estáveis
- **Hashing dinâmico**



# Arquivo Físico e Arquivo Lógico

---

- **Arquivo Físico:** sequência de bytes armazenada no disco
- **Arquivo Lógico:** arquivo como visto pelo aplicativo que o acessa
- **Associação arquivo físico – arquivo lógico:** iniciada pelo aplicativo, gerenciada pelo S.O.



# Arquivo Físico e Arquivo Lógico

---

- **Arquivo Físico:** conjunto de bytes no disco, geralmente agrupados em setores de dados. Gerenciado pelo sistema operacional
- **Arquivo Lógico:** modo como a linguagem de programação enxerga os dados. Uma sequência de bytes, eventualmente organizados em registros ou outra estrutura lógica.



# Arquivo Físico e Arquivo Lógico

---

- Arquivo lógico é como usar um **telefone** para se conectar com qualquer pessoa
  - Não precisamos saber onde a pessoa está, mas podemos falar com ela
- O arquivo lógico pode se relacionar a um **arquivo em disco** ou a outros **dispositivos de E/S**



## Exemplo: Associação entre Arquivo Físico e Arquivo Lógico

---

- Em C: (associa e abre para leitura)

```
file *p;
```

```
if ((p=fopen("meuarq.dat", "r"))==NULL)
```

```
    printf("erro...")
```

```
else ...
```



# Abertura de Arquivos

---

- Arquivo novo (p/ escrita) ou arquivo já existente (p/ leitura ou escrita)...



# Abertura de Arquivos

---

- Em C
  - Comandos  
fopen
  - Parâmetros especiais indicam o modo de abertura



# Função fopen

---

**fd=fopen(<filename>,<flags>)**

- <filename>: nome do arquivo a ser aberto
- <flags>: controlam o modo de abertura
  - **"r"** apenas leitura modo texto; o arquivo precisa existir
  - **"w"** cria arquivo vazio para escrita modo texto (se já existe, é apagado)
  - **"a"** adiciona (append) texto no final do arquivo (se arquivo não existe, cria)
  - **"r+"** Abre arquivo para leitura e escrita modo texto
  - **"w+"** Cria arquivo vazio para leitura e escrita modo texto
  - **"a+"** Abre arquivo para leitura e adição modo texto
  - **rb, wb, ab, rb+, wb+, ab+** : análogo para modo binário



# 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. Interessante para:
  - Prevenir contra interrupção
  - Liberar as estruturas associadas ao arquivo para outros arquivos



# Exemplo: fechamento de arquivos

---

C:

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

```
.....
```

```
fclose(fd)
```



# Leitura e Escrita Modo Texto

---

- C: Funções da linguagem
  - **fgets(<cadeia>, <nro\_caracteres>, <fd>)**
  - **fputs(<cadeia>, <fd>)**
    - dados lidos/escritos como *strings*
    - *Não se especifica formato*



# Leitura e Escrita Modo Texto

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
    char str[80];
    FILE *fp;
    if ((fp=fopen("teste.txt","w"))==NULL) {
        printf("Arquivo não encontrado");
        exit(1);
    }
    do {
        printf("Entre com uma string: ");
        gets(str);
        fputs(str,fp);
    } while (strcmp(str,"fim"));
    fclose(fp);
    return(0);
}
```

Lê da console até a cadeia "fim" e grava no arquivo



# Leitura e Escrita Modo Texto

---

- C: Funções da linguagem
  - **fscanf(fd,formato,argumentos)**
  - **fprintf(fd,formato,argumentos)**
    - dados lidos/escritos de modo formatado
    - Especifica formato (+ flexível)



# Leitura e Escrita Modo Texto

```
#include <stdio.h>
int main() {
    FILE *f, *g;
    char c, d;
    f=fopen("arq1.txt", "r");
    g=fopen("arq2.txt", "w");
    switch (fscanf(f, "%c %c", &c, &d)) {
        case 1: fprintf(g, "%c", c); break; // leu um valor
        case 2: fprintf(g, "%c %c", c, d); break; // leu dois valores
        default: if ferror(f)
                    printf("problemas na leitura do arquivo");
                    break;
    }
    fclose(f);
    fclose(g);
    return(0);
}
```

Lê 2 caracteres de um arquivo e grava em outro



# Leitura e Escrita Caracteres

---

- C: Funções da linguagem
  - `fgetc()`
  - `fputc()`
    - dados lidos/escritos um caracter por vez



# Leitura e Escrita Modo Binário

---

- **fread(<dest-address>,<nro\_bytes>,<contador>,fd)**
- **fwrite(<dest-address>,<nro\_bytes>,<contador>,fd)**
  - dados lidos/escritos como registros ou blocos de bytes
  - modo binário



# Leitura e Escrita Modo Binário

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *arq;
    float f=25.5, f1;
    int i=12, i1;
    if ((arq=fopen("teste.txt", "wb+"))==NULL)
        exit(1);
    fwrite(&f, sizeof(float), 1, arq);//escreve um bloco de tamanho de float
a partir do endereço de f
    fwrite(&i, sizeof(int), 1, arq);
    rewind(arq);
    fread(&f1, sizeof(float), 1, arq);
    fread(&i1, sizeof(int), 1, arq);
    fclose(arq);
    return(0);
}
```

Grava real e inteiro num  
arquivo, e depois os lê



# Fim de Arquivo

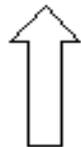
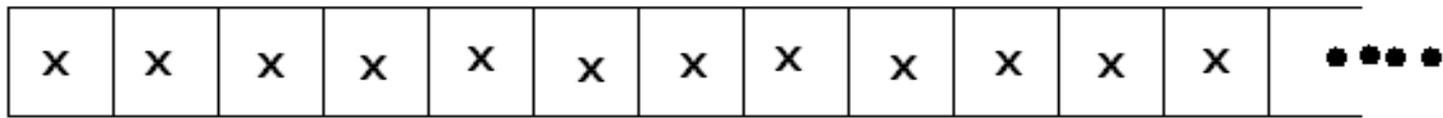
---

- Ponteiro de arquivo: controla o próximo byte a ser lido
- C:
  - **feof()**
    - Retorna 1, se fim de arquivo; 0, c.c.



# O ponteiro no arquivo lógico

---



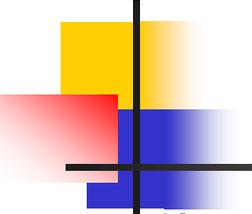
**Ponteiro indicando a posição atual  
x - um byte qualquer**



# 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):** acesso envolve o posicionamento do ponteiro em um byte ou registro arbitrário



# Seeking

---

- Ação de mover o ponteiro para uma certa posição no arquivo
  - **pos=fseek(fd, byte-offset, origin)**
    - Função retorna a posição final do ponteiro
    - **byte-offset** – deslocamento, em bytes, a partir de *origin*
    - Origin:
      - 0 – início do arquivo    1 – posição corrente    2 – final do arquivo
    - Retorna 0 quando bem sucedida



# Seeking

---

```
#include <stdio.h>
struct dados {
    char c;
    int x;
    float y;
} item;

#define TAM sizeof(struct dados);
int main() {
    FILE *pont;
    pont=fopen("arquivo.txt", "r");
    if (fseek(pont, 9*TAM, 0)) //posiciona no início do 10º. Reg.
        printf("Erro na busca");
    else fread(&item, TAM, 1, pont); // lê registro atual
    return(0);
}
```

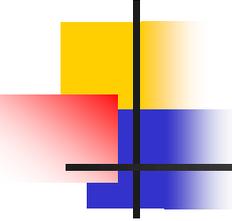
Lê 10º. registro de  
arquivo



# Bufferização

---

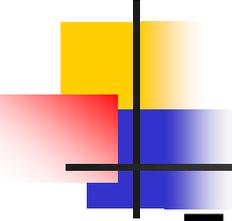
- Toda operação de I/O é 'bufferizada'
  - Buffer: I/O de dispositivos exceto discos (teclado, vídeo, etc.)
  - Memória Cache: I/O discos 256K, 640K
  - Os bytes passam por uma 'memória de transferência' de tamanho fixo e de acesso otimizado, de maneira a serem transferidos em blocos
    - Por quê?



# Bufferização

---

- Qual o tamanho dos blocos de leitura/escrita?
  - Depende do SO e da organização do disco
  - Sistema de arquivo: gerencia a manipulação de dados no disco, determinando como arquivos podem ser gravados, alterados, nomeados ou apagados
    - Ex. No Windows, é determinado pela FAT – *File Allocation Table* (FAT16, FAT32 ou NTFS)



# Leitura e Escrita

---

- Exercício

- Escrever um programa em C que

- Declare uma estrutura com o nome e a nota de um aluno
- Leia do usuário os nomes e as notas de uma turma de graduação com 50 alunos (assuma que cada aluno tem um número identificador de 1 a 50)
- Escreva em um arquivo todos os dados lidos
- Recupere o nome e a nota de um aluno de número especificado pelo usuário