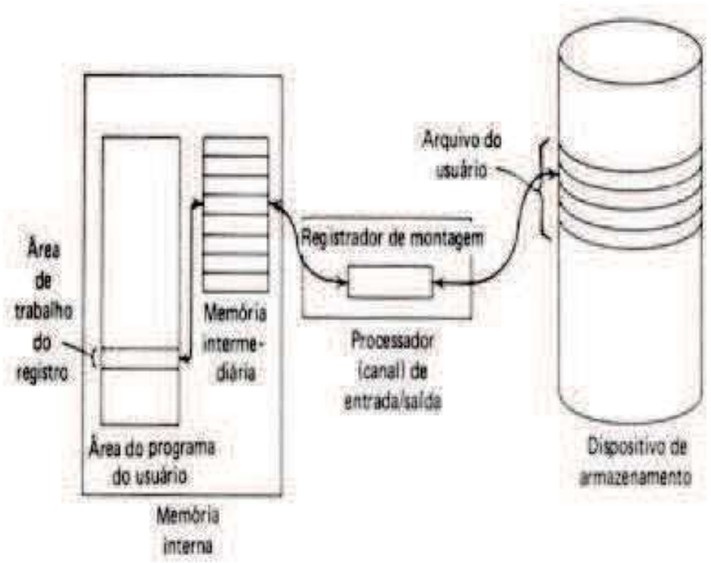
Revisão de Conceitos

- Arquivos em C
- Estruturas de Dados
- Programação Orientada a Objetos (C++)
- Documentação de Códigos (Doxygen)

Projeto de Arquivos - O que trataremos hoje?

Conceitos envolvendo: SO, I/O, blocagem, registros lógicos, registros físicos, abstração de arquivos, tipos de dispositivos de armazenamento, ...



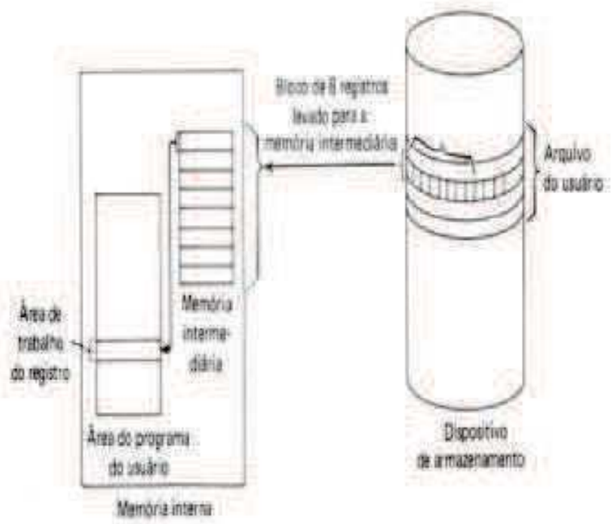


Figura 1.3 Bloco de oito registros lógicos movido para a memória intermediária e depois, um a um, para a área de trabalho do registro.

Fonte: [Claybrook, 1983]

Projeto de Arquivos - Definição

A definição de *projeto*, de acordo com o dicionário Michaelis é

"Plano para a realização de um ato"

Por analogia, podemos dizer que *projeto de arquivos* significa

"plano/estudo visando a implementação mais adequada de arquivos de dados".

Nos sistemas computacionais de interesse em PRA esse estudo estará associado ao gerenciamento e organização de arquivos de dados.

Projeto de Arquivos

O gerenciamento de arquivos a ser implementada depende:

1. do tipo de dispositivo de armazenamento, suas características físicas e operacionais;
2. da necessidades da aplicação que utiliza os dados armazenados;
3. das características funcionais não apenas dos processadores de entrada e saída mas também do próprio SO.

Projeto de Arquivos - Programador de Aplicação

Mais ainda...

Na visão do programador de aplicação o foco desse projeto está na manipulação das estruturas de dados para a atualização, remoção, inserção, ordenação, intercalação, cópia, etc, de arquivos de maneira eficiente...

Um arquivo, nesse caso, é uma entidade lógica “vista” através de estruturas de dados que o acessam através de uma abstração de sua real condição conforme os dispositivos que o armazenam.

Essa abstração é fornecida através do sistema operacional...

Projeto de Arquivos - Programador de Aplicação

Entidades são objetos sobre os quais se armazenam informações. Ex: empregados de uma Cia

Entidades possuem propriedades caraterísticas chamadas atributos. Ex: nome, endereço, idade, matrícula,...

Funcionário:
- Nome
- Matrícula
- Endereço
- Estado civil
- ...

Projeto de Arquivos - Programador de Aplicação

Um registro (ou registro lógico) é um conjunto de pares

$$< atributo, valor >$$

Ex: Nome: João, idade: 55, matrícula: 1234...

Um conjunto de registros de uma classe de entidades é armazenado na memória principal em arrays/tabelas e em memória secundária em arquivos.

Projeto de Arquivos - Programador de Aplicação

Arquivo coleção de estruturas de dados agregando valores heterogêneos (registros) ou não (caracteres).

A linguagem de programação expressa o nível lógico-conceitual (armazenamento/recuperação) na forma de registros e arquivos.

O sistema operacional mapeia os registros e arquivos em dispositivos reais de armazenamento [Ferraz, 2003].

Projeto de Arquivos - Programador de Sistema

O programador de sistema, por sua vez, lida com aspectos de projeto tais como:

- Sistema de blocagem,
- *caching*,
- diretórios,
- canais de I/O,
- rotinas de baixo nível que permitem acesso a registros independentemente do dispositivo,
- etc...

Projeto de Arquivos - Programador de Sistema

Começando pela visão do programador de sistema:

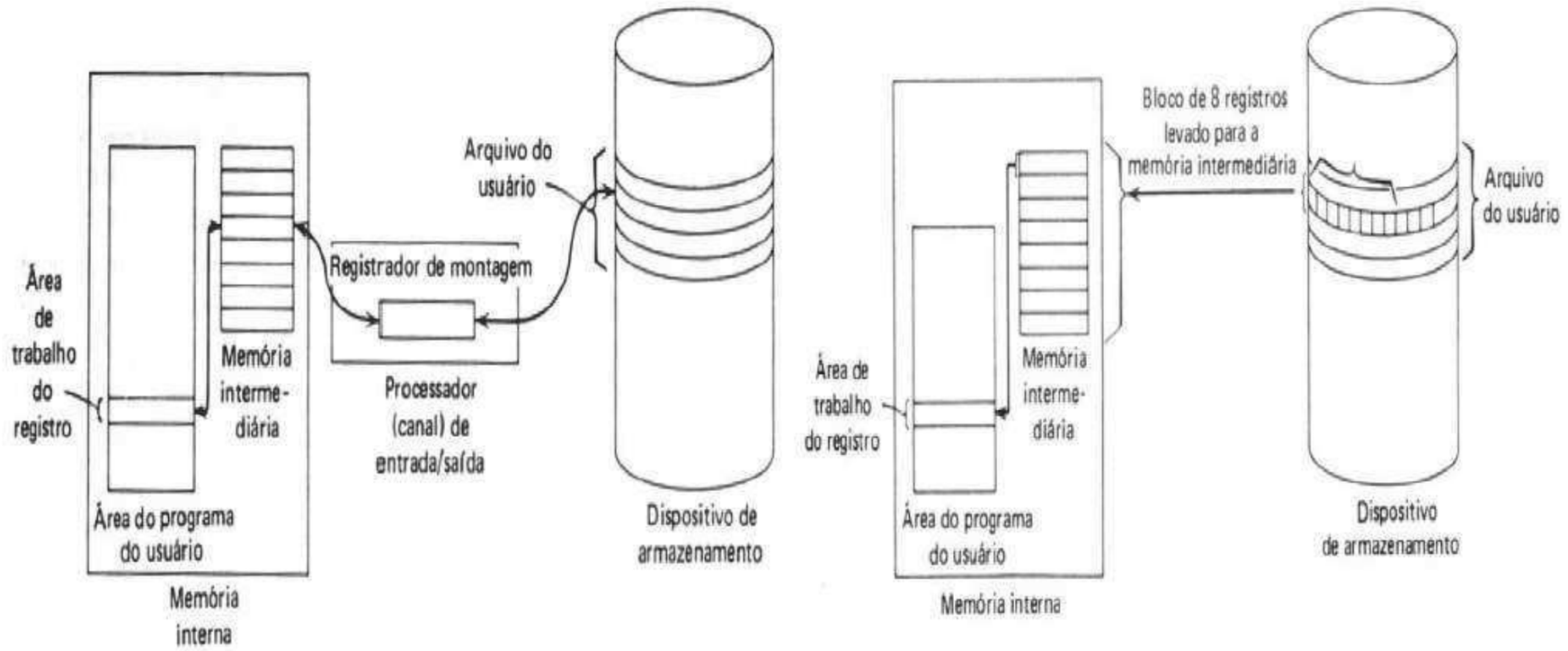


Figura 1.2 Fluxo de dados entre a memória interna e o armazenamento externo.

Figura 1.3 Bloco de oito registros lógicos movido para a memória intermediária e depois, um a um, para a área de trabalho do registro.

Figura 1: Comunicação Memória Interna X Memória Externa

Fonte: [Claybrook, 1983]

Projeto de Arquivos - Programador de Sistema

Programadores de sistema devem criar/prover abstrações de dispositivos de armazenamento.

Essas abstrações são manipuladas pelo programador de aplicação.

Através dessas abstrações o programador de aplicação pode se referir a diferentes e complexos dispositivos de armazenamento de forma sintaticamente simples utilizando as linguagens de programação:

```
FILE *fp=NULL;
...
fp = fopen(nomeArq,"w+b");
rewind(fp);
fread(&aux,sizeof(reg),1,fp);
...
```

Figura 2: Exemplo em C

Projeto de Arquivos - Programador de Sistema

Os dispositivos de armazenamento secundário mais comuns são:

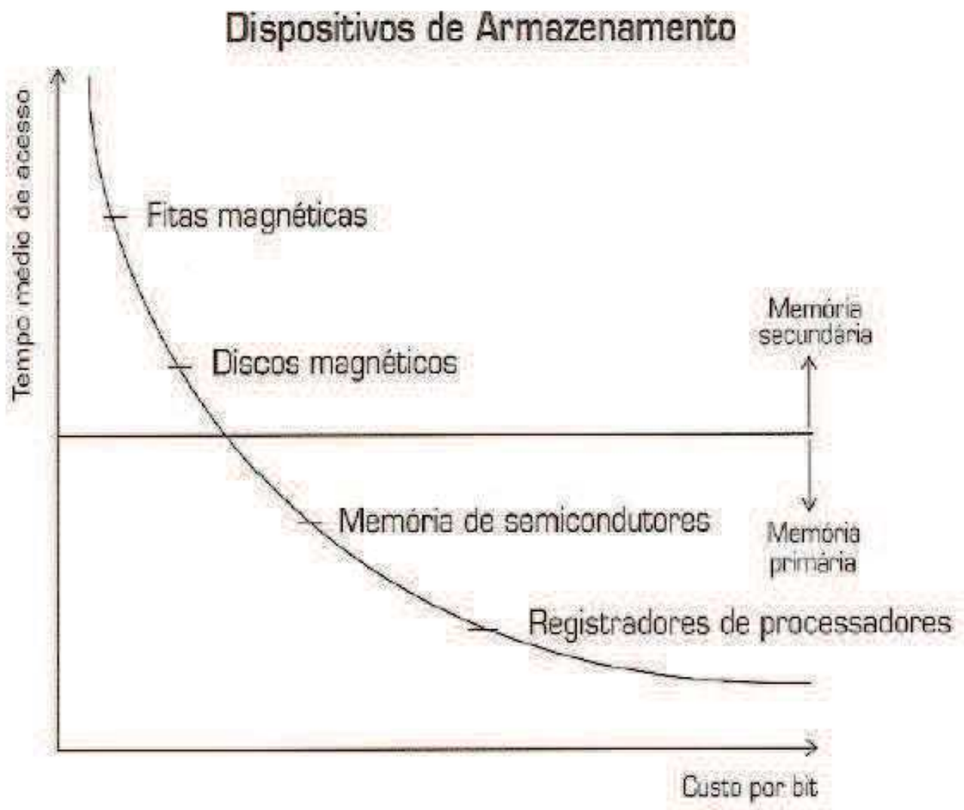
- Fitas,
- Discos magnéticos,
- Discos ópticos e ultimamente
- os Solid-State Drives.

Independente do dispositivo, é importante analisar:

1. Capacidade de armazenamento;
2. **Portabilidade:** quando se tratar de dispositivos removíveis;
3. Custo relativo;
4. Tamanho de registro (dados contíguos) endereçável;
5. **Método de acesso:** seqüencial, direto (aleatório)
6. Velocidade de transferência da (ou para a) memoria principal
7. Tempo de posicionamento para mover a cabeça r/w até o a trilha contendo o registro alvo
8. **Latência:** tempo de retardo rotacional para discos ou o tempo de partida para a fita alcançar velocidade operacional.

Projeto de Arquivos - Programador de Sistema

Dispositivos de armazenamento secundário são mais baratos, mais lentos e de maior capacidade do que os dispositivos de memória interna (principal):



Projeto de Arquivos - Programador de Sistema

Fitas em Rolos:

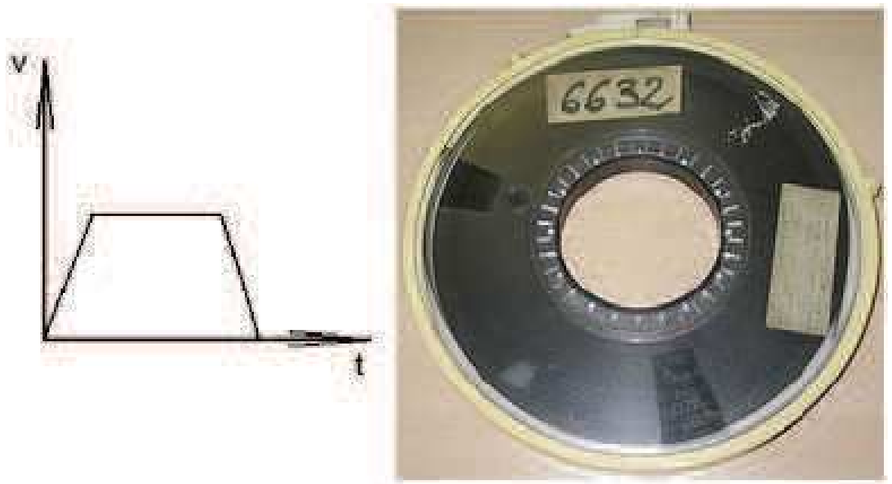


Figura 3: Acionador Start-Stop

Fitas em Cartuchos:

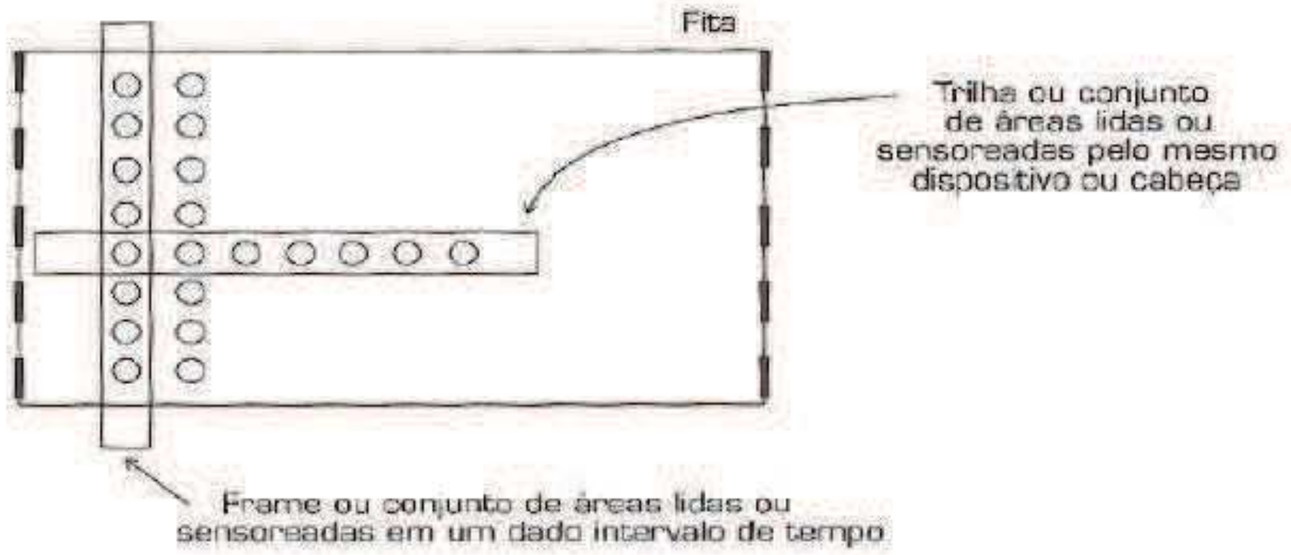
Figura 4: Acionador Streaming

Fontes: [Ferraz, 2003] e http://en.wikipedia.org/wiki/Magnetic_tape_data_storage

Projeto de Arquivos - Programador de Sistema

Fitas em Rolos:

Blocos de dados e intervalos entre blocos.



Projeto de Arquivos - Programador de Sistema

Fitas em Rolos - Cálculo do Comprimento S_T [Claybrook, 1983]:

$$comprimentoTotal = numeroTotaldeBlocos \times comprimentoDeUmBloco \rightarrow$$

$$S_T = \frac{N_L}{BF}\left(IRG + \frac{BF \times LRL}{DEN}\right)$$

onde:

- S_T: comprimento total da fita para o arquivo (polegadas)
- N_L: quantidade de registros lógicos do arquivo
- BF: fator de blocagem (quantos registros lógicos cabem em um registro físico/bloco físico)
- IRG: separador de registros (polegadas)
- LRL: comprimento do registro lógico (caracteres)
- DEN: densidade (bits ou caracteres por polegada)

Projeto de Arquivos - Programador de Sistema

Fitas em Rolos - Exemplo de Cálculo do Comprimento S_T:
Dados:

N_L: 10.000 registros lógicos

BF: 100 (registros por bloco)

IRG: 0.75”

LRL: 160 caracteres

DEN: 800 cpi

$$S_T = \frac{10000}{100}\left(0.75 + \frac{100 \times 160}{800}\right) = 2075 \; \textit{polegadas} \cong 52.7 \; \textit{metros}$$

Projeto de Arquivos - Programador de Sistema

Fitas em Rolos - Cálculo do Tempo para ler o arquivo T_T [Claybrook, 1983]:

$tempoTotal = numTotaldeBlocos \times (tempoParaUltrapassarIRG + tempoParaLerUmBloco) \rightarrow$

$$T_T = \frac{N_L}{BF}\left(T_A + \frac{BF \times LRL}{SPD \times DEN}\right)$$

onde:

T_T: tempo total de leitura para o arquivo

SPD: velocidade de leitura da fita (em polegadas/segundo)

$SPD \times DEN$: velocidade de transferência

T_A: tempo de partida/parada ou tempo para ultrapassar IRG (em milissegundos)

Projeto de Arquivos - Programador de Sistema

Fitas em Rolos - Exemplo de Cálculo do Tempo de Leitura T_T:
Dados:

N_L: 10.000 registros lógicos

BF: 100 (registros por bloco)

T_A: 12,6 ms

LRL: 160 caracteres

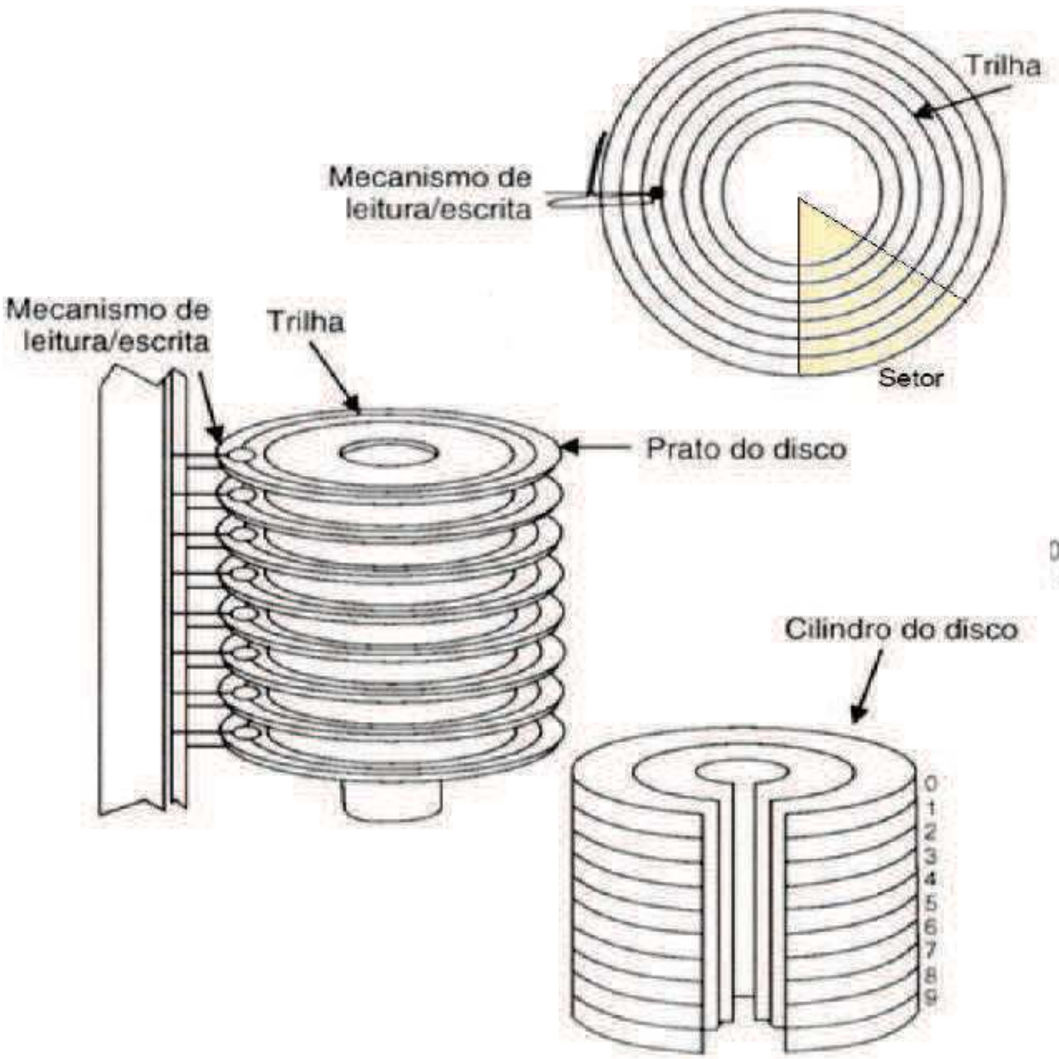
DEN: 800 cpi

SPD: 75 pol/seg

$$T_T = \frac{10000}{100}\left(0.75 + \frac{100 \times 160}{75 \times 800}\right) \cong 27,9 \; \textit{segundos}$$

Projeto de Arquivos - Programador de Sistema

Discos Magnéticos (HD)



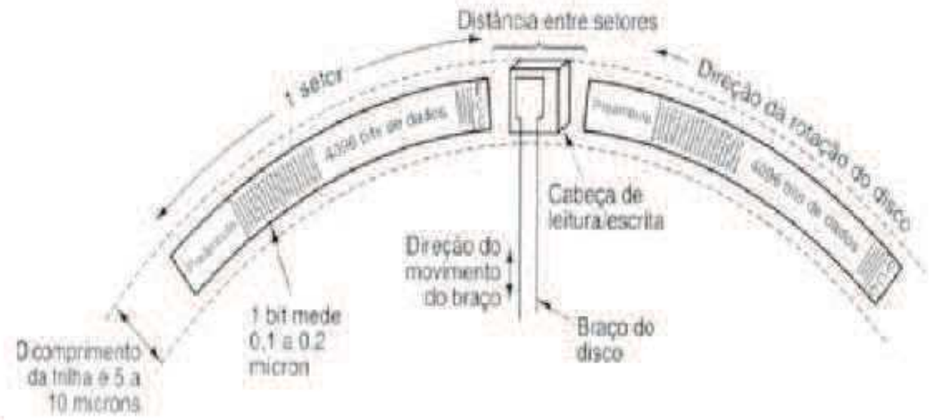


Fig. 2.19 Parte da trilha de um disco. Estão ilustrados dois setores.

Projeto de Arquivos - Programador de Sistema

Discos Magnéticos (HD)
O ideal é que o arquivo ocupe setores/trilhas e cilindros contíguos, caso contrário o disco estará fragmentado e exigirá um tempo maior para posicionamento do cabeçote r/w nas diversas regiões onde se fragmentou o arquivo.

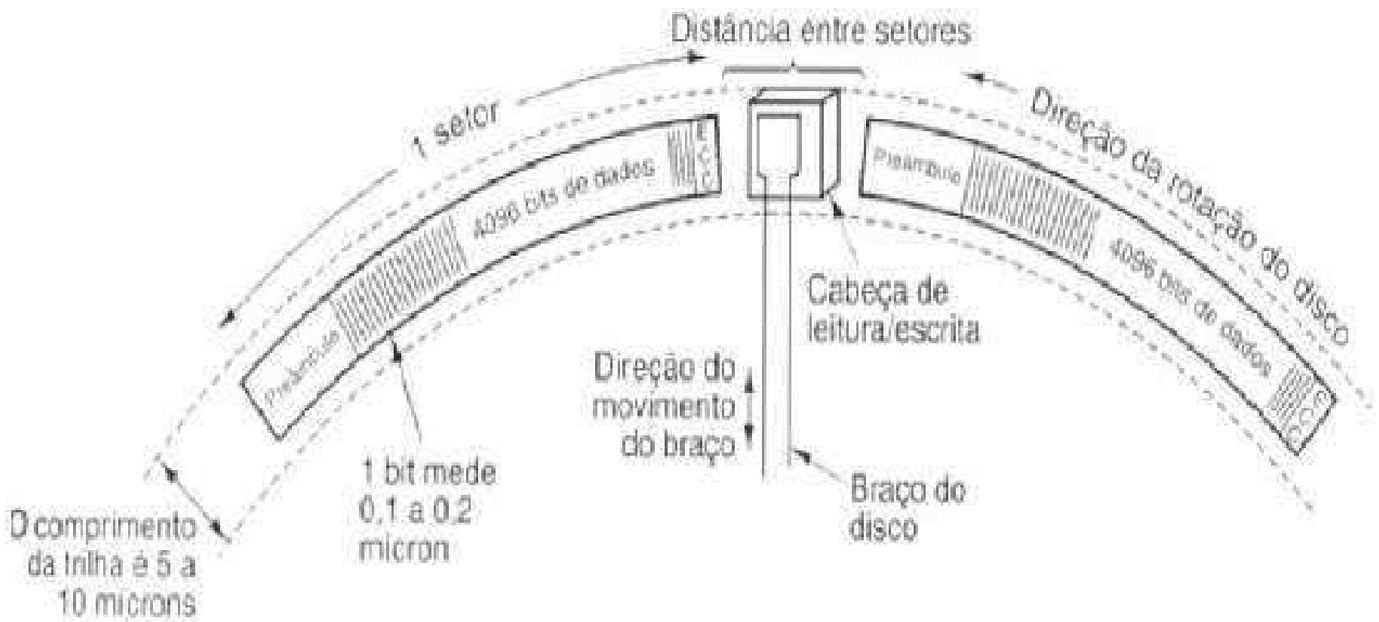


Figura 5: Parte da trilha de um disco. Estão ilustrados dois setores.

Projeto de Arquivos - Programador de Sistema

Discos Magnéticos - Cálculo do Número de Trilhas para armazenar um arquivo N_T [Claybrook, 1983]:

$$N_T = \frac{N_B}{N_{B/T}} \quad N_B = \frac{N_L}{BF} \quad N_{B/T} = \left\lfloor \frac{T_{CAP}}{B_S} \right\rfloor \quad B_S = G_S + NDI + BF \times LRL$$

onde:

- N_B: número de blocos por arquivo
- N_L: Número de registros lógicos de um arquivo
- BF: Fator de bloco (blocagem)
- $N_{B/T}$: Número de blocos por trilha
- T_{CAP}: Capacidade de trilha
- B_S: Tamanho do bloco em caracteres
- G_S: Espaço total do separador de bloco
- NDI: Dados do sistema por bloco (caracteres)
- LRL: Comprimento do Registro Lógico (caracteres)

Projeto de Arquivos - Programador de Sistema

Discos Magnéticos - Capacidade de Armazenamento:

$$T_{CAP} = N_{S/T} \times N_{B/S}$$

$$T_C = N_{T/C} \times T_{CAP}$$

$$T_{HD} = N_C \times T_C$$

onde:

- T_{HD}: capacidade do HD
- T_{CAP}: capacidade da trilha
- $N_{S/T}$: número de setores por trilha
- $N_{B/S}$: número bytes por setor
- T_C: Capacidade do cilindro
- $N_{T/C}$: número de trilhas por cilindro
- N_C: número de cilindros

Projeto de Arquivos - Programador de Sistema

$$T_{ACCESS} = T_{SEEK} + T_{LAT} + T_{DT}$$

- Tempo de Acesso

  T_{SEEK}: tempo de posicionamento
  T_{LAT}: (tempo de latência), tempo para que o bloco (alvo) passe sob a cabeça R/W e possa ser processada a transferência de dados
  T_{DT}: tempo de transferência

  O tempo para mudar entre cabeças de R/W é desprezível e não entra na conta.

- Em um disco não fragmentado, ler um arquivo seqüencialmente é mais rápido, pois o acesso seqüencial minimiza o tempo de busca e de latência.
- Para uma leitura aleatória (fragmentada) , quase todos os acessos terão tempo de busca, tempo de latência e tempo de transferência.

Projeto de Arquivos - Programador de Sistema

Considere uma unidade de disco magnético com as seguintes características :

- Setor : 512 bytes
- Taxa de transferência : 20 Mbytes/seg
- Tempo de seek médio : 8 ms
- Tempo latência médio : 4 ms
- Capacidade por trilha : 85 Kbytes
- Capacidade total : 9100 Mbytes
- N° de pratos/superfícies : 10/20
- Responda:
  1. Qual é o número de cilindros do disco ?
  2. Quantos registros de 128 bytes podem ser armazenados em 1 cilindro do disco ?
  3. Calcule o tempo gasto para a leitura seqüencial de 50.000 setores.
  4. Calcule o tempo gasto para a leitura aleatória de 50.000 setores.

Projeto de Arquivos - Programador de Sistema

- **Qual é o número de cilindros do disco ?**

$$\frac{9100 \text{ MB}}{20 \text{ superfícies}} = 455 \text{ MB/superfície}$$

$$455 \times 1024 \times \frac{1024 \text{ bytes}}{85 \times 1024 \text{ bytes/trilha}} = 5.481 \text{ trilhas/superfície}$$

O número de cilindros no disco é igual ao número de trilhas por superfície. Portanto, há 5.481 cilindros no disco.

Projeto de Arquivos - Programador de Sistema

- **Quantos registros de 128 bytes podem ser armazenados em 1 cilindro do disco ?**

$$(85 \times 1024 \text{ bytes/trilha}) \times 20 \text{ trilhas/cilindro} = 1.740.800 \text{ bytes}$$

$$\text{Portanto cabem } \frac{1.740.800 \text{ bytes}}{128 \text{ bytes/registro}} = 13.600 \text{ registros em um cilindro.}$$

Projeto de Arquivos - Programador de Sistema

- **Calcule o tempo gasto para a leitura seqüencial de 50.000 setores.**
  - A leitura seqüencial envolve um tempo de seek, um tempo de latência e o tempo gasto com a transferência. Tempo de seek: 8 ms. Tempo de latência: 4ms
  - Tempo de transferência:

$$50.000 \text{ setores} \times 512 \text{ bytes/setor} = \frac{25.600.000 \text{ bytes}}{20 \times 1024 \times 1024 \text{ bytes/seg}} = 1,22 \text{ seg}$$

  - Tempo total: $0,008 + 0,004 + 1,22 \cong 1,23$ seg

Projeto de Arquivos - Programador de Sistema

- **Calcule o tempo gasto para a leitura aleatória de 50.000 setores.**
  - A leitura de cada setor requer um tempo de seek, um tempo de latência e o tempo de transferência. Dessa maneira, serão necessários 50.000 seeks e 50.000 latências no total, além do tempo de transferência.

  - Tempo total: $50.000 \times (0,008 + 0,004 \text{ seg}) + 1,22 \text{ seg} = 601,22 seg$

Revisão sobre Arquivos em Linguagem C

Arquivos em C

Arquivos Texto: os dados são armazenados sem uma representação fixa, ou seja, meramente como uma sequência de caracteres.

Isto permite que programas editores de texto (Notepad, Word, etc.) possam abri-los e editá-los como qualquer outro arquivo, sem a necessidade de se utilizar o programa de onde o arquivo foi originalmente produzido.

Este tipo de arquivo requer que o programa que o esteja manipulando seja responsável por "dar sentido" aos caracteres lidos/gravados (*Parsing*). Por exemplo: os primeiros 10 caracteres de cada linha representam um nome, os próximos 3 um código numérico e assim por diante.

Arquivos Binários: os dados são armazenados como sequências de bytes formados (variáveis, estruturas, etc.).

Dado que há um formato específico para o armazenamento da informação, apenas o programa de origem é capaz de ler os dados armazendos no arquivo. Outra vantagem esta no fato de que dados complexos podem ser lidos com apenas um acesso à disco (p.ex. ler um estrutura composta por vários campos simultaneamente).

Arquivos em C

Abrir um arquivo em C significa criar uma *stream* e conectá-la ao arquivo em disco. Em C, **fopen( )** é a função que abre (ou cria) arquivos, seu protótipo encontra-se abaixo e seu uso necessita da inclusão de **stdio.h**.

FILE fopen(const char* filename, const char* mode);

filename indica o nome do arquivo a ser aberto/criado

mode mode associado à *stream*

Se a operação de abertura transcorre sem problemas `fopen( )` devolve um ponteiro de referência que será usado para manipular o arquivo, caso haja algum erro um NULL será devolvido.

Arquivos em C

Os modos de abertura do arquivo **TEXTO** podem ser:

Valor	Descrição
"r" ou "rt"	Abre arquivo texto apenas para leitura e posiciona o ponteiro no início do arquivo. Se o arquivo não existe (previamente), fopen devolverá NULL.
"w" ou "wt"	Abre arquivo texto apenas para escrita. Se o arquivo não existe ele será criado. **Se o arquivo já existe ele será aberto para escrita, porém seu conteúdo será apagado.** Posiciona o ponteiro no início do arquivo.
"a" ou "at"	Abre arquivo texto para anexação, nesse caso **só será possível acrescentar dados a partir do ponto de abertura que sempre ocorrerá no final do arquivo**, portanto não há possibilidade de perda dos que já existam no arquivo. Se o arquivo já existe ele será aberto para anexação e o ponteiro de arquivo será posicionado no final deste (EOF – *End Of File*). Se não existe, ele será criado e o ponteiro posicionado no inicio deste.
"r+" ou "r+t"	Abre arquivo texto para leitura e escrita e posiciona o ponteiro no início do arquivo. Se o arquivo não existe (previamente), fopen devolverá NULL.
"w+" ou "w+t"	Abre arquivo texto para escrita e leitura. Se o arquivo não existe ele será criado. Se já existe ele será aberto, porém seu conteúdo será apagado. Posiciona o ponteiro no início do arquivo.
"a+" ou "a+t"	Abre arquivo texto para anexação (será possível acrescentar dados , ou seja, escrever dados sem perda dos que já existam no arquivo) e para leitura. Será possível retornar o ponteiro inclusive para posições anteriores à da abertura, porém para estas posições só será possível e efetuar operações de leitura. Se o arquivo já existe ele será aberto para anexação e o ponteiro de arquivo será posicionado no final deste (EOF). Se não existe, ele será criado e o ponteiro posicionado no inicio deste.

OBS.: Para abrir arquivos **binários** basta substituir o símbolo "t" nos casos acima por "b".

Arquivos em C - Exemplos

```
1   #include "stdio.h"
2    ....
3   main(void) {
4       FILE * fp;
5       char  filename[]  = "arq.txt";
6
7       if ((fp  = fopen(filename,  "w+")) == NULL) {
8           printf ("Erro  na  abertura  do arquivo");
9           exit (0);  }
10
11          .....
12  }
```

Arquivos em C - Exemplos

```
1   #include "stdio.h"
2    ....
3   main(void) {
4       FILE * fp;
5       char  filename[]  = "c:\\teste\\arq.txt",
6              modo[] = "w+";
7
8       if ((fp  = fopen(filename,  modo)) == NULL) {
9           printf ("Erro  na  abertura  do arquivo");
10          exit (0);  }
11
12        .....
13  }
```

Arquivos em C - Exemplos

```
1   #include "stdio.h"
2   #define MSG_ERR "Qtde Invalida de Parametros"
3    ....
4   main(int argc, char * argv[]) {
5       FILE * fp;
6
7       if (argc < 2) {
8           printf (MSG_ERR);
9           exit (0); }
10
11      if ((fp = fopen(argv[1], "w+")) == NULL) {
12          printf ("Erro na abertura do arquivo");
13          exit (0); }
14
15       .....
16  }
```

Arquivos em C

Stream: fluxo de movimentação de dados entre a memória principal e a secundária.

Buffer: memória lógica auxiliar associada a um *stream*.

- Para esvaziar o *buffer* corrente usa-se o comando:

int fflush(FILE * fp)

- Para fechar o *stream* corrente usa-se o comando:

int fclose(FILE * fp)

Arquivos em C

Acesso Sequencial vs Aleatório

1. Arquivos em C funciona como uma fita lógica;
2. Eles são manipulados através de um *apontador* que indica a posição (em bytes) do ponto exato sendo acessado na 'fita";
3. Início do arquivo é indicado pela posição zero;
4. Arquivos recém-criados tem comprimento zero e o ponteiro sempre indicará a posição zero;
5. Para arquivos pré-existentes recém abertos, o ponteiro indicará a posição definida pelo modo de abertura do mesmo (`a, r, w, ...`);
6. Tanto a leitura quanto a escrita em arquivo, levam em consideração a posição do apontador;
7. A posição do apontador é atualizada após cada operação em função da quantidade de bytes lidos/escritos;

Arquivos em C

Exemplo:

- Ao abrir um arquivo texto para leitura, solicitou-se a leitura de 10 caracteres;
- Cada caracter tem tamanho 1 byte;

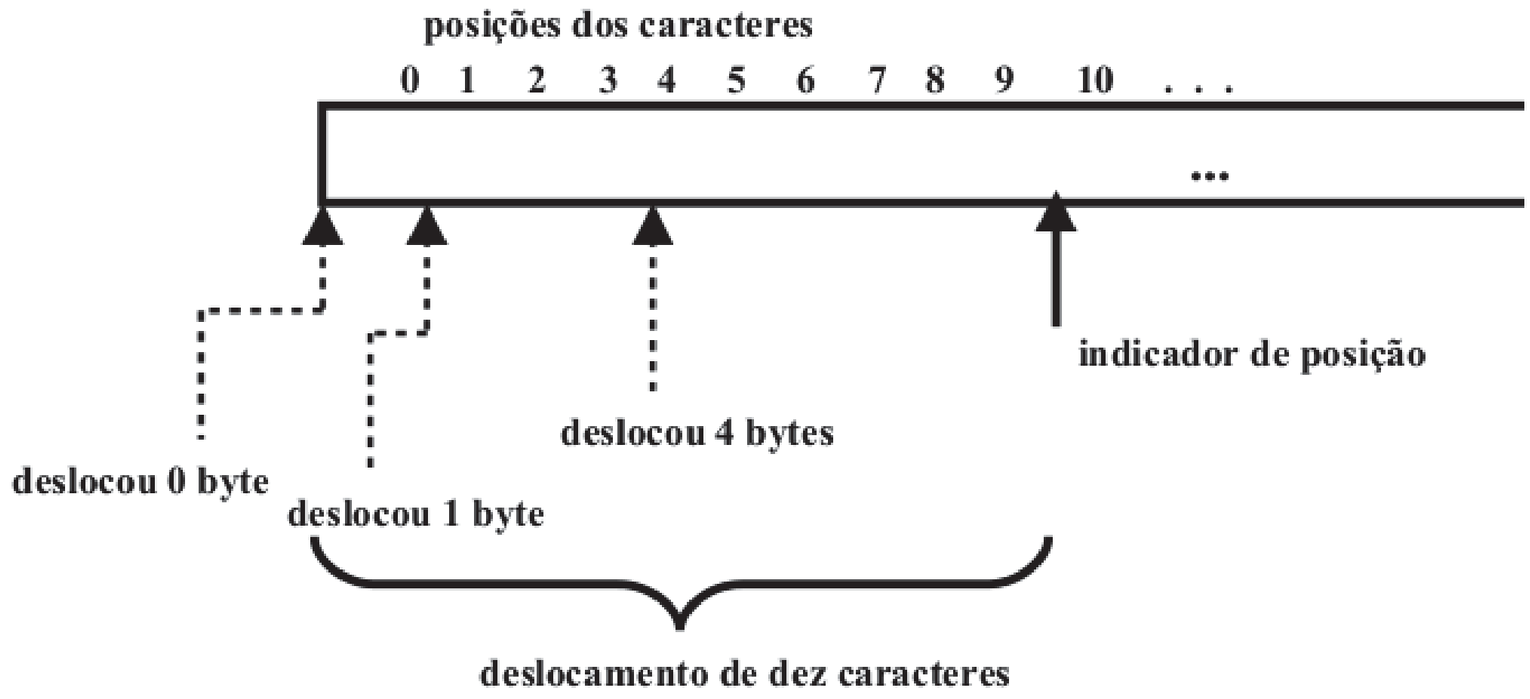


Figura 6: Situação de um arquivo após operação de leitura de 10 caracteres

Arquivos em C - Acesso Aleatório

- Para reposicionar o apontador de leitura novamente no início do arquivo, usa-se:

void rewind(FILE * fp)

- Para reposicionar o apontador de leitura para uma posição específica, usa-se:

int fseek(FILE * fp, long numbytes, int origin)

onde:

1. ponteiro para o arquivo a ser manipulado
2. quantidade de bytes a serem deslocados
3. ponto de referência, em relação ao qual, o deslocamento é realizado. Possíveis valores:

Macro	Valor	Descrição
SEEK_SET	0	Mover à partir do início do arquivo
SEEK_CUR	1	Mover à partir da posição atual
SEEK_END	2	Mover à partir do final do arquivo

Arquivos em C - Acesso Aleatório

- Para determinar a posição atual (em bytes) do apontador de leitura em relação ao início do arquivo. Preferencialmente utilizado para arquivos binários. Uso:

int ftell(FILE * fp)

Observação: Se um arquivo for aberto em modo *append* (“a”), o comando *ftell* retorna zero se, antes disso, não for executado um

fseek(fp, 0L, SEEK_END)

Arquivos em C - Acesso Aleatório

Exemplos de uso:

- Para posicionar o apontador no nono registro do arquivo:

fseek(fp, 9 * sizeof(struct list_type), SEEK_SET);

- Para posicionar o apontador no início do arquivo:

fseek(fp, 0L, SEEK_SET);

- Para re-posicionar o apontador no início do arquivo:

rewind(fp);

ou

fseek(fp, - ftell(fp), 1);

- Para posicionar o apontador no final do arquivo:

fseek(fp, 0L, SEEK_END);

Arquivos em C - I/O (modo texto)

Comandos de leitura e/ou gravação em arquivos texto:

- **int putc(int ch, FILE * fp)** escreve um caracter *ch* no arquivo *fp*. Se bem sucedido, a função retorna o próprio caracter, caso contrário retorna `EOF (End-Of-File)`.
- **int getc(FILE * fp)** lê o caracter posicionado sob o apontador de leitura. Retorna `EOF` se o fim de arquivo for alcançado.
- **int fputs(const char * str, FILE * fp)** grava uma cadeia de caracteres *str* em *fp*. Retorna `EOF` caso um erro de gravação ocorra. A função também traduz o caracter '`\n`' para os símbolos `CRLF = carriage return / line feed`.
- **int fgets(char * str, int tam, FILE * fp)** lê uma cadeia de caracteres em *str* (terminada em '`\0`'), de um comprimento máximo *tam* (será menor que tam se um fim de linha for lido antes). A função retorna `NULL` caso ocorra um erro de leitura ou `EOF` seja encontrado.

Arquivos em C - I/O (modo texto)

Comandos de leitura e/ou gravação em arquivos texto:

- Para uso de formato específico para o processo de leitura/escrita utilizam-se os comandos:

  int fprintf(FILE * fp, const char * string_formato, <lista de variáveis...>)

  int fscanf(FILE * fp, const char * string_formato, <lista de variáveis...>)

Arquivos em C - I/O (modo texto)

Para se verificar se o fim de arquivo foi alcançado:

```
while( ( c = fgetc( fp ) ) != EOF )
...
```

Arquivos em C - I/O (modo binário)

Comandos de leitura e/ou gravação em arquivos binários:

- **int fread(void * buf, int size, int count, FILE * fp)** efetua a leitura para um *buffer* de dados (*buf*), de uma certa quantidade de dados (*count*) de um certo tipo de dados de tamanho *size* (em bytes) a partir de um arquivo *fp*.
- **int fwrite(void * buf, int size, int count, FILE * fp)** efetua a gravação de uma certa quantidade (*count*) de um *buffer* de dados (*buf*), de um certo tipo de dados de tamanho *size* (em bytes) para um arquivo *fp*.

Arquivos em C - I/O (modo binário)

Exemplo:

```
1   typedef struct {
2       int i;
3       char str[30];
4   }reg;
5
6   main(void)
7   {
8       FILE *fp;
9       int count, size;
10      reg registro, aux;
11      char nome_arq[] = "teste.bin";
12      registro.i = 100;
13      strcpy(registro.str," valor ");
14
15      if ((fp = fopen(nome_arq,"w+b")) == NULL)
16      { printf("Erro na abertura do arquivo"); exit(0); }
17
18      size = sizeof(reg);
19      count = 1;
20      if ((fwrite(&registro, size, count, fp)) < count)
21      { puts("Erro na operacao de escrita"); exit(0); }
22
23      rewind(fp);
24      fread(&aux,sizeof(reg),1,fp);
25      printf(" valores lidos: %i, %s ", aux.i,aux.str);
26      fclose(fp);
27  }
```

Arquivos em C - I/O (modo binário)

Mais duas funções úteis

- **int rename(const char * oldname, const char * newname)** renomeia o arquivo chamado *oldname* para *newname*. Atenção!: o arquivo oldname não pode estar em uso.
  A função devolve zero caso a operação resulte em sucesso ou diferente de zero caso ocorra erro.
- **int remove(const char * filename)** apaga o arquivo definido por *filename*. Atenção!: o arquivo filename não pode estar em uso.
  A função devolve zero caso a operação resulte em sucesso ou diferente de zero caso ocorra erro.

Organização de Arquivos

- Um banco de dados é armazenado em uma coleção de arquivos. Cada arquivo é uma sequência de registros. Um registro é uma sequência de campos;
- Uma chave é uma sequência de um ou mais campos;
- O valor de uma chave-primária identifica um único registro. O valor de uma chave-secundária identifica um conjunto de registros;
- Os registros podem ser de tamanho fixo ou variável;

Org. de Arquivos - Registros de Tamanho Fixo

- Mais fáceis de implementar
- Cada arquivo possui registros de um tipo particular
- Arquivos diferentes são usados por relações (conceito de banco de dados) diferentes
- Abordagem simples: "concepção vetorial"
  - Armazene o registro i começando no byte $n * (i \check{} 1)$, onde n é corresponde ao tamanho de cada registro.
  - O acesso é simples porém, a não ser que o buffer/bloco seja de tamanho múltiplo de um registro, pode ocorrer o cruzamento da fronteira do bloco
- Alternativas para remoção do registro i:
  - mova registros $i + 1, \ldots, n$ para $i, \ldots, n \check{} 1$
  - mova apenas o registro n para i
  - Não mova nada. Utilize uma lista encadeada de "registros livres": *Free List*

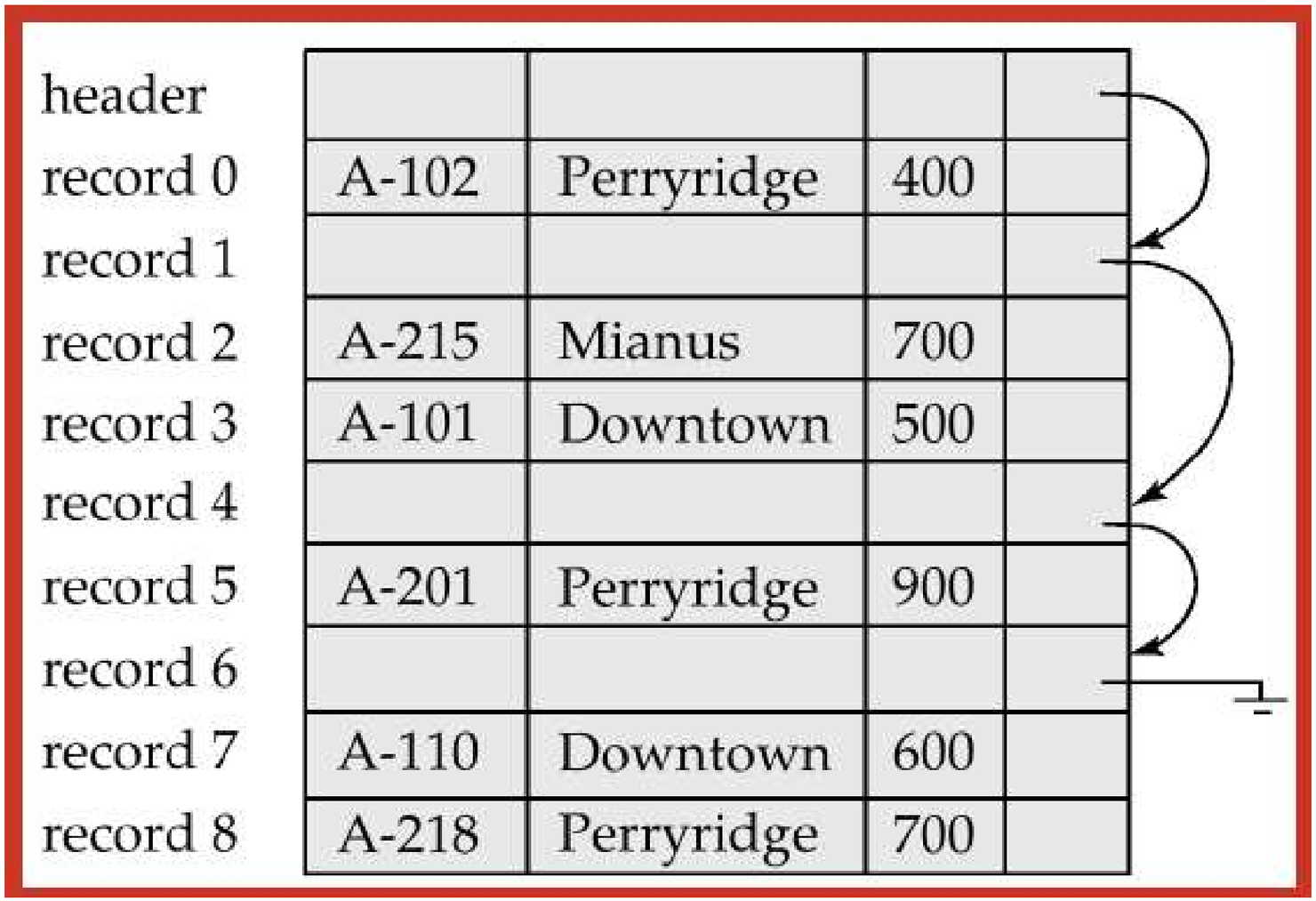
Org. de Arquivos - Registros de Tamanho Fixo

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

Org. de Arquivos - Reg. de Tamanho Fixo - *Free list*

- O endereço do primeiro registro deletado é armazenado no header.
- Use o primeiro registro para armazenar o endereço do segundo registro deletado e assim por diante.
- Imagine que os endereços armazenados são como ponteiros porém apontando para uma localização no arquivo

Org. de Arquivos - Reg. de Tamanho Fixo - *Free list*



Org. de Arquivos - Registros de Tamanho Variável

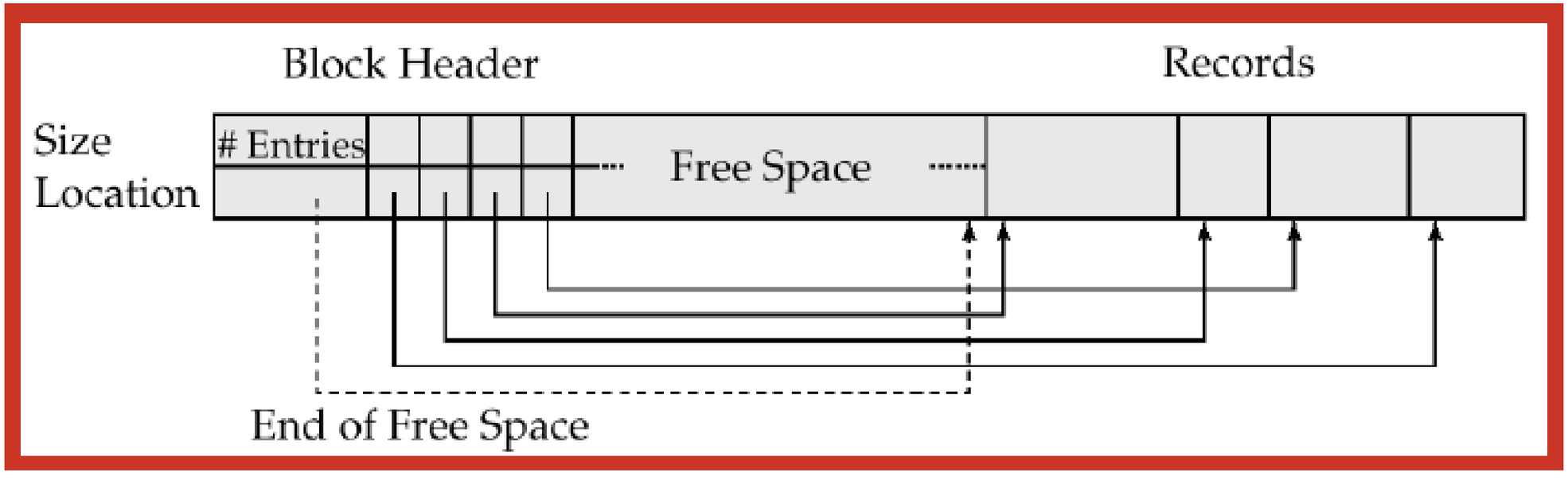
- Esse tipo de registro surge em decorrência de vários fatores:
  - Armazenamento de tipos variados de registros em um arquivo.
  - O tipo de registro permite comprimento variável de um ou mais campos.
  - O registro permite campos repetidos.
- Representação na forma de cadeia de bytes (**Byte string**)
  - Utilize um caracter de controle *end-of-record* ($\perp$) para marcar o fim de cada registro
  - Problemas com a remoção do registro: não é eficiente a reutilização do espaço o que provoca fragmentação no disco
  - Problemas se há crescimento (maior) do registro: preciso realocar o registro em operação delicada.

Org. de Arquivos - Reg. de Tamanho Variável - *Byte string*

0	Perryridge	A-102	400	A-201	900	A-218	700	⊥
1	Round Hill	A-305	350	⊥				
2	Mianus	A-215	700	⊥				
3	Downtown	A-101	500	A-110	600	⊥		
4	Redwood	A-222	700	⊥				
5	Brighton	A-217	750	⊥				

Org. de Arquivos - Reg. de Tamanho Variável - *Slotted Page Structure*

- O descritor da **Slotted page** contém:
  - Número de entradas de registro
  - Fim do espaço livre no bloco
  - Um vetor contendo a localização e tamanho de cada registro



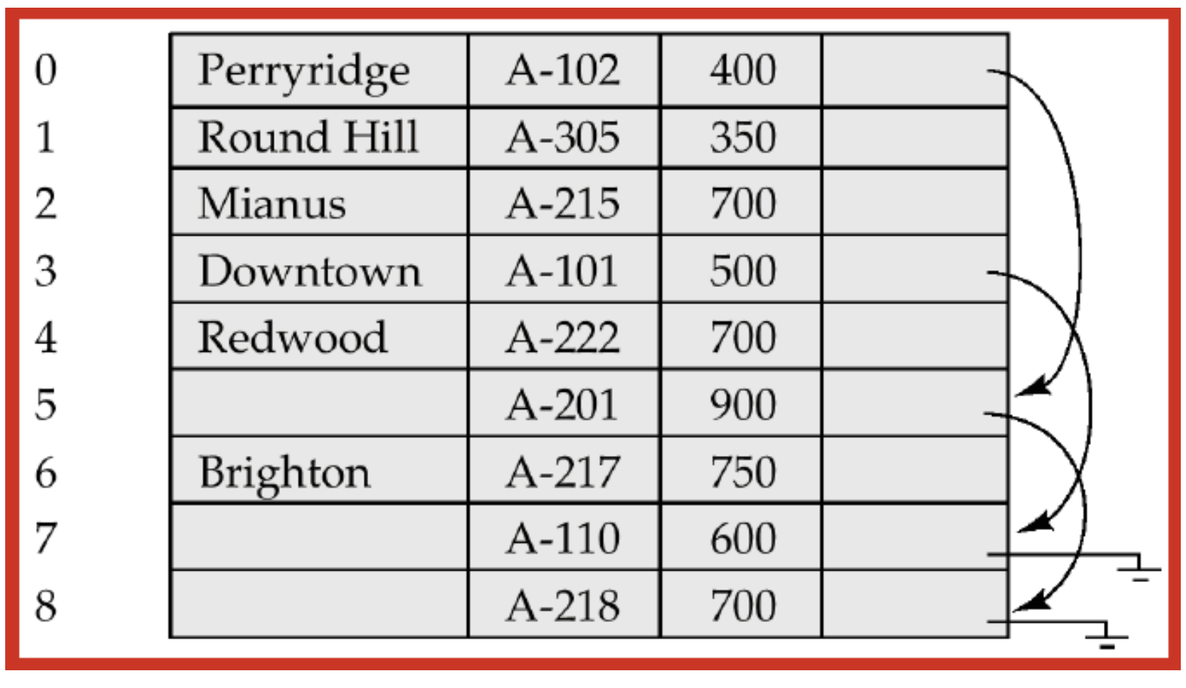
Org. de Arquivos - Registros de Tamanho Variável

- Representação por meio de vários registros de tamanho fixo:
  - Espaço reservado
  - Ponteiros
  - **Espaço reservado** – utiliza o registro de máximo tamanho como um tamanho fixo; registros menores que este recebem marcas *end-of-record* no espaço não utilizado.

0	Perryridge	A-102	400	A-201	900	A-218	700
1	Round Hill	A-305	350	⊥	⊥	⊥	⊥
2	Mianus	A-215	700	⊥	⊥	⊥	⊥
3	Downtown	A-101	500	A-110	600	⊥	⊥
4	Redwood	A-222	700	⊥	⊥	⊥	⊥
5	Brighton	A-217	750	⊥	⊥	⊥	⊥

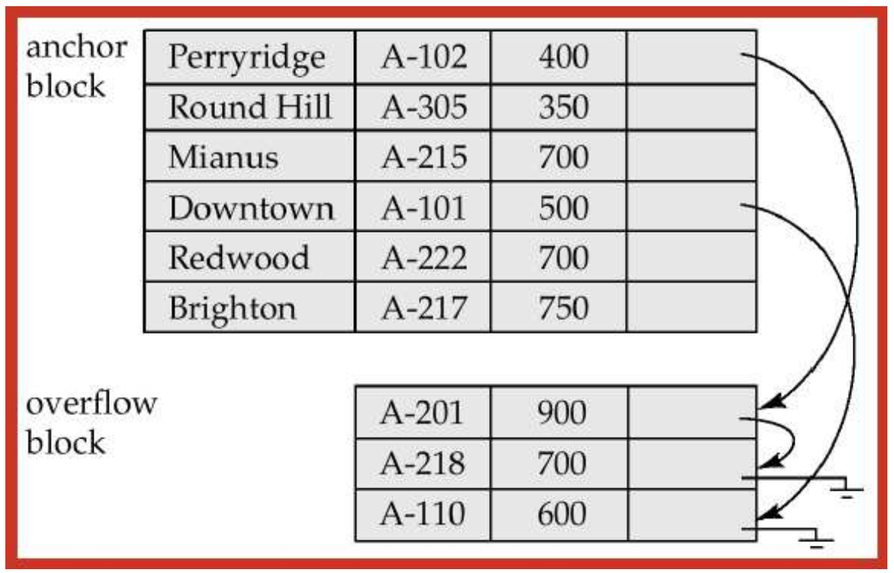
Org. de Arquivos - Reg. de Tamanho Variável - *Ponteiro*

- Método do Ponteiro
  - Um registro de tamanho variável é representado por uma lista encadeada de registros de tamanho fixo.
  - É interessante quando é desconhecido o registro de tamanho máximo



Org. de Arquivos - Reg. de Tamanho Variável - *Ponteiro*

- Desvantagem: desperdiça espaço em todos os registros encadeados, exceto o primeiro.
- A solução é permitir dois tipos de blocos:
  Bloco âncora contém o primeiro registro da cadeia
  Bloco de Overflow contém o demais registros que não aqueles que são os primeiros de uma cadeia



Organização de Registros no Arquivo

- **Heap** – O registro pode ser colocado em qualquer local livre no arquivo
- **Sequencial** – Armazena registros em ordem sequencial baseado no valor de uma chave de pesquisa
- **Hashing** – Uma *hash-function* é computada sobre algum atributo de registro; o resultado especifica em qual bloco do arquivo o registro será colocado
- Registros de cada relação podem ser armazenados em arquivos separados. Porém, em uma organização **clustering** os registros de relações diferentes podem ser armazenadas no mesmo arquivo
  - Motivação: armazenar registros em um mesmo bloco minimiza I/Os

Avaliação #1 - Arquivos

Título: Trabalho Prático #1 - Construção de uma Base de Dados

Objetivo: Construir uma base de dados inicial que servirá como argumento de entrada para todos os demais trabalhos de indexação de arquivos a serem vistos na disciplina.

Forma de Entrega: Deve ser entregue todos os códigos-fonte (devidamente documentados/comentados), bem como deve ser preparada uma apresentação que inclui uma demonstração de funcionamento do sistema.
Deve ainda ser entregue uma cópia do arquivo construído previamente pelo sistema, que contenha (pelo menos) 1 GB de tamanho em disco.

Prazo de Entrega: 2 semanas

Avaliação #1 - Arquivos

Critérios:
1. Equipes de 3 alunos
2. Cada equipe recebe um tema diferente para a construção dos registros da base de dados
3. O programa a ser desenvolvido deve permitir ao usuário:
   - 3.1 Escolher a quantidade de registros a ser gerada (pelo processo estocástico) ou o tamanho físico do arquivo a ser gerado;
   - 3.2 Permitir a gravação de registros através de paginação (tamanho definido pelo usuário);
   - 3.3 Permitir a recuperação de registros através de paginação (tamanho definido pelo usuário) e de leitura sequencial;
   - 3.4 Exibir ao final do processo o tempo gasto para o processamento.

Avaliação #1 - Arquivos

Temas:

- **Tema 1** - **Escola**: cadastro de alunos contendo: nome do aluno, data de nascimento (dd/mm/aaaa), n° de matrícula, lista de disciplinas matriculadas (quantidade indeterminada);
- **Tema 2** - **Diário de Classe**: relatório acadêmico contendo: nome do aluno, n° de matrícula, notas das provas realizadas (quantidade indeterminada), n° de faltas, média arimética e situação acadêmica (aprovado, exame ou reprovado);
- **Tema 3** - **Loja**: cadastro de clientes contendo: nome do cliente, endereço, telefone, data de nascimento (dd/mm/aaaa), código identificador, lista de itens (códigos dos produtos) comprados (qtde. indeterminada);

Avaliação #1 - Arquivos

Temas:

- **Tema 4** - **Vendas**: listagem de vendas de uma loja: código do cliente, código do vendedor, data da venda (dd/mm/aaaa), valor da venda (R$), listagem dos itens comprados (qtde. indeterminada);
- **Tema 5** - **Fluxo de Caixa**: relatório de operações realizadas: código sequencial da operação, indicador de compra/venda (C ou V), valor da operação, data da operação (dd/mm/aaaa);
- **Tema 6** - **Resultados do futebol**: relatório com os resultados do campeonato: nome do time *A*, nome do time *B*, placar do jogo, data de realização da partida (dd/mm/aaaa), público pagante, local do jogo;

Avaliação #1 - Arquivos

Temas:

- **Tema 7** - **Maratona**: relatório de chegada dos corredores: código do corredor, nome do corredor, tempo de duração da corrida (hh:mm:ss:ms), data da corrida (dd/mm/aaaa);
- **Tema 8** - **Previsão do Tempo**: dados da previsão do tempo: nome do local, data da coleta dos dados (dd/mm/aaaa), hora da coleta dos dados (hh:mm:ss:ms), temperatura medida (° C);
- **Tema 9** - **Imobiliária**: listagem de imóveis para locação: tipo do imóvel (casa ou apto), endereço, n° de quartos, preço do aluguel (R$), data em que o imóvel ficou disponível (dd/mm/aaaa);

Avaliação #1 - Arquivos

Temas:

- **Tema 10** - **Hospital**: relatório médico dos pacientes: código sequencial da operação, nome do médico, nome do paciente, data da internação (dd/mm/aaaa), código do motivo da internação, lista de sintomas;
- **Tema 11** - **ACM ICPC**: equipes da maratona de programação: nome do componente #1, nome do componente #2, nome do componente #3, nome da equipe, n° de balões obtidos, n° de tentativas falsas;