

# Organização de Arquivos (part 1)

**Prof. Dr. Lucas C. Ribas**

**Disciplina:** Estrutura de Dados II

Departamento de Ciências de Computação e Estatística



UNIVERSIDADE ESTADUAL PAULISTA  
“JÚLIO DE MESQUITA FILHO”



IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO

# Tópicos da aula



- Introdução
- Organização em campos
- Organização em registros
- Acesso a registros



- Ao construir uma estrutura de arquivos, estamos impondo uma organização aos dados
- Qual a diferença entre os termos **stream** e **arquivo**?
  - caminho por meio do qual os programas podem enviar uma seqüência de bytes de uma fonte até um destino. •
  - normalmente estão associados a arquivos, mas uma fonte ou destino também podem ser: teclado, mouse, memória, vídeo ou impressora.
  - Streams lidam apenas com entrada ou saída de bytes



- Faça um programa em C que

- Leia do usuário os seguintes dados de 10 pessoas: nome, idade, número de filhos
- Escreva em um arquivo os dados lidos
- Leia do arquivo os dados escritos

- Alguns comandos:

- `f=fopen(nome_arquivo, modo_de_abertura)`
- `fclose(f)`
- `fscanf(f, formato, argumentos)`
- `fprintf(f, formato, argumentos)`
- `fseek(f, byte-offset, origem)`



```
#include <stdio.h>
#define TAM 2

struct aluno {
    char nome[20];
    int idade;
    int nota;
};

int main() {
    int i;
    struct aluno a;
    FILE *f;

    printf("Lendo dados\n\n");
    f=fopen("saida.txt","w");
    for (i=0; i<TAM; i++) {
        printf("Entre com o nome do aluno: ");
        scanf("%s",a.nome);
        printf("Entre com a idade: ");
        scanf("%d",&a.idade);
        printf("Entre com a nota: ");
        scanf("%d",&a.nota);
        printf("\n");
        fprintf(f,"%s",a.nome);
        fprintf(f,"%d",a.idade);
        fprintf(f,"%d",a.nota);
    }
    fclose(f);

    printf("Dados no arquivo\n\n");
    f=fopen("saida.txt","r");
    for (i=0; i<TAM; i++) {
        fscanf(f,"%s",a.nome);
        printf("Nome: %s\n",a.nome);
        fscanf(f,"%d",&a.idade);
        printf("Idade: %d\n",a.idade);
        fscanf(f,"%d",&a.nota);
        printf("Nota: %d\n\n",a.nota);
    }
    fclose(f);

    return(0);
}
```

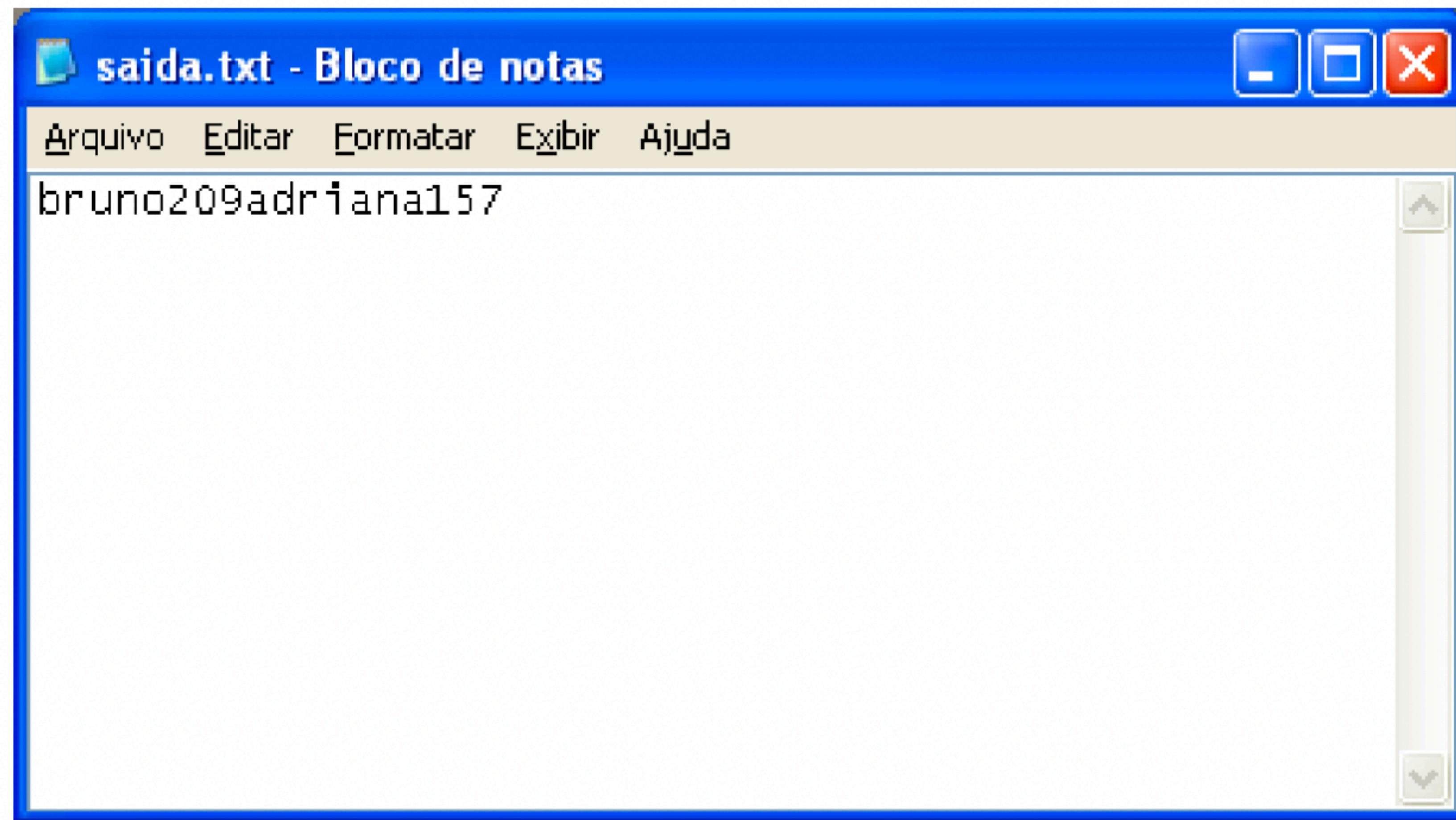
O que esse programa  
vai imprimir?

# Exemplo de execução



```
Lendo dados  
Entre com o nome do aluno: bruno  
Entre com a idade: 20  
Entre com a nota: 9  
  
Entre com o nome do aluno: adriana  
Entre com a idade: 15  
Entre com a nota: 7  
  
Dados no arquivo  
Nome: bruno209adriana15?  
Idade: 15  
Nota: 7  
  
Nome: bruno209adriana15?  
Idade: 15  
Nota: 7
```

# Exemplo de execução





- Informações em arquivos são, em geral, organizadas em **campos e registros**
- **Conceitos Lógicos**
  - Não necessariamente correspondem a uma organização física
- Dependendo de como a informação é mantida no arquivo, **campos lógicos sequer podem ser recuperados**
- **Exemplo**
  - Suponha que desejamos armazenar em um arquivo os nomes e endereços de várias pessoas
  - Suponha que decidimos representar os dados como uma sequência de bytes (sem delimitadores, contadores, etc.)
  - AmesJohn123 MapleStillwaterOK74075MasonAlan90 EastgateAdaOK74820



- **Não há como recuperar** porções individuais (nome ou endereço)
  - Perde-se a **integridade** das unidades fundamentais de organização dos dados
- Os dados são agregados de caracteres com significado próprio
  - Tais agregados são chamados **campos (fields)**

# Organização em campos



## ○ Campo

- Menor unidade lógica de informação em um arquivo
  - Uma noção lógica (ferramenta conceitual), não corresponde necessariamente a um conceito físico
- 
- Existem várias maneiras de organizar um arquivo mantendo a identidade dos campos
  - A organização anterior não proporciona isso

# Métodos para organização em campos



- Comprimento fixo
- Indicador de comprimento
- Delimitadores
- Uso de *tags* (etiquetas)

# Campos com tamanho fixo



- Cada campo ocupa no arquivo um **tamanho fixo**, pré-determinado
- O fato do tamanho ser conhecido garante que é possível recuperar cada **campo**
- Como?

Maria	Rua 1	123	São Carlos
João	Rua A	255	Rio Claro
Pedro	Rua 10	56	Rib. Preto

```
struct {  
    char last[10];  
    char first[10];  
    char city[15];  
    char state[2];  
    char zip[9];  
} set_of_fields;
```

# Campos com tamanho fixo



- Quais as **desvantagens** desta abordagem?
- O espaço alocado (e não usado) aumenta desnecessariamente o tamanho do arquivo (**desperdício**)
  - Solução **inapropriada** quando se tem uma grande quantidade de **dados com tamanho variável**
  - Razoável apenas se o comprimento dos campos é realmente fixo ou apresenta pouca variação

# Campos com indicador de comprimento



- O tamanho de cada campo é armazenado imediatamente antes do dado
  - Se o tamanho do campo é inferior a 256 bytes, o espaço necessário para armazenar a informação de comprimento é um único byte
- Desvantagens desta abordagem?

05	Maria	05	Rua	103	123	10	São Carlos
04	João	05	Rua A	03	255	09	Rio Claro
05	Pedro	06	Rua	10	0256	10	Rib. Preto

# Campos separados por delimitadores



- Caractere(s) especial(ais) (que não fazem parte do dado) são escolhido(s) para ser(em) inserido(s) ao final de cada campo
  - Ex.: para o campo nome pode-se utilizar /, tab, #, etc...
  - Espaços em branco não servem na maioria dos casos

Maria Rua 1 123 São Carlos
João Rua A 255 Rio Claro
Pedro Rua 10 56 Rib. Preto

# Uso de *tag* do tipo “keyword=value”



- Vantagem: o campo fornece **informação semântica** sobre si próprio
  - Fica mais fácil identificar o conteúdo do arquivo
  - Fica mais fácil identificar campos perdidos
- Desvantagem: as *keywords* podem ocupar uma porção significativa do arquivo

```
Nome=Maria|Endereço=Rua 1|Número=123|Cidade=São Carlos|
Nome=João|Endereço=Rua A|Número=255|Cidade=Rio Claro|
Nome=Pedro|Endereço=Rua 10|Número=56|Cidade=Rib. Preto|
```

# Organização em registros



## ◎ Registro: um conjunto de campos agrupados

- Arquivo representado em um **nível de organização mais alto**
  - É um outro nível de organização imposto aos dados com o objetivo de preservar o significado
- Assim como o conceito de campo, um registro é uma **ferramenta conceitual**, que não existe necessariamente no sentido físico

## ◎ Métodos para organização em registros:

- Tamanho fixo
- Número fixo de campos
- Indicador de tamanho
- Uso de índice
- Utilizar delimitadores

# Registros de tamanho fixo



- Analogamente ao conceito de **campos de tamanho fixo**, assume que todos os registros têm o **mesmo tamanho**, com **campos de tamanho fixo ou não**
  - Um dos métodos mais comuns de organização de arquivos

Registro de tamanho fixo e campos de tamanho fixo:

Maria	Rua 1	123	São Carlos
João	Rua A	255	Rio Claro
Pedro	Rua 10	56	Rib. Preto

Registro de tamanho fixo e campos de tamanho variável:

Maria Rua 1 123 São Carlos	← Espaço vazio →
João Rua A 255 Rio Claro	← Espaço vazio →
Pedro Rua 10 56 Rib. Preto	← Espaço vazio →

# Registros com número fixo de campos



- Ao invés de especificar que cada registro contém um tamanho fixo, podemos especificar um **número fixo de campos**
  - O tamanho do registro é **variável**
  - Neste caso, os campos seriam separados por **delimitadores**

Registro com número fixo de campos:

Maria|Rua 1|123|São Carlos|João|Rua A|255|Rio Claro|Pedro|Rua  
10|56|Rib. Preto|

# Indicador de tamanho para registros



- O indicador que precede o registro fornece o seu **tamanho total**
  - Os campos são separados internamente por **delimitadores**
  - Boa solução para registros de tamanho variável

Registro iniciados por indicador de tamanho:

28Maria|Rua 1|123|São Carlos|25João|Rua A|255|Rio Claro|27Pedro|Rua  
10|56|Rib. Preto|



- Um **índice externo** poderia indicar o deslocamento de cada registro relativo ao início do arquivo
  - Pode ser utilizado também para calcular o **tamanho dos registros**
  - Os campos seriam separados por **delimitadores**

Arquivos de dados + arquivo de índices:

Dados: Maria|Rua 1|123|São Carlos|João|Rua A|255|Rio Claro|Pedro|Rua  
10|56|Ribeirão Preto|  
Índice: 00 29 44

A diagram illustrating the relationship between the data fields and the index values. Red arrows point from the index values to specific fields in the data row. The first arrow points to the first field ('Maria'), the second to the third ('São Carlos'), and the third to the eighth field ('Ribeirão').



- Separar os registros com **delimitadores** análogos aos de fim de campo
  - O delimitador de campos é mantido, sendo que o método combina os dois delimitadores
  - Note que delimitar fim de campo é diferente de delimitar fim de registro

Registro delimitado por marcador (#):

Maria|Rua 1|123|São Carlos|#João|Rua A|255|Rio Claro|#Pedro|Rua 10|56|Rib.  
Preto|

# Acesso a registros



- Arquivos organizados por registros

- **Como buscar um registro específico?**

- Cada registro poderia ter uma identificação única
      - Aluno de número X
      - Livro de código Y



- Uma **chave** (key) está associada a um registro e permite a sua recuperação
  - O conceito de chave é também uma ferramenta conceitual importante



- Uma **chave primária** é, por definição, a chave utilizada para identificar unicamente um registro
  - Exemplo: matrícula UNESP, CPF, RG
  - Sobrenome, por outro lado, não é uma boa escolha para chave primária
- Uma **chave secundária**, tipicamente, não identifica unicamente um registro, e pode ser utilizada para buscas simultâneas por vários registros
  - Todos os “**Silvas**” que moram em **São Paulo**, por exemplo



- O ideal é que exista uma **relação um a um entre chave e registro**
- Se isso não acontecer, é necessário fornecer uma **maneira** do usuário decidir qual dos registros é o que interessa

# Escolha da Chave Primária



- Preferencialmente, a **chave primária deve ser "dataless"**, isto é, não deve ter um significado associado, e não deve **mudar nunca** (outra razão para não ter significado)
- Uma mudança de significado pode implicar na mudança do valor da chave, o que invalidaria referências já existentes baseadas na chave antiga

# Forma canônica da chave



- **Formas canônicas** para as chaves: uma única representação da chave que conforme com uma regra
  - "Ana", "ANA", ou "ana" devem levar ao mesmo registro
- Ex: a regra pode ser 'todos os caracteres maiúsculos'
  - Nesse caso a forma canônica da chave será ANA



- Na pesquisa em RAM, normalmente adotamos como medida do trabalho necessário **o número de comparações** efetuadas para obter o resultado da pesquisa
- Na pesquisa em arquivos, o acesso a disco é a operação mais cara e, portanto, **o número de acessos a disco** efetuados é adotado como medida do trabalho necessário para obter o resultado
- **Mecanismo de avaliação do custo associado ao método:** contagem do número de **chamadas à função de baixo nível READ**
- Assumimos que:
  - Cada chamada a READ lê 1 registro e requer um seek
  - Todas as chamadas a READ têm o mesmo custo

# Busca sequencial



- Busca pelo registro que tem uma determinada chave em um arquivo
  - Lê o arquivo registro a registro, em busca de um registro contendo um certo valor de chave
- Uma busca por um registro em um arquivo com 2.000 registros
  - Requer, em média, 1.000 leituras
    - 1 leitura se for o primeiro registro, 2.000 se for o último e, portanto, 1000 em média
  - No pior caso, o trabalho necessário para buscar um registro em um arquivo de tamanho  $n$  utilizando busca sequencial é  $O(n)$

# Blocagem de Registros



- A operação **seek** é lenta
- A **transferência dos dados** do disco para a RAM é **relativamente rápida...**
  - apesar de muito mais lenta que uma transferência de dados em RAM
- O custo de buscar e ler um registro, e depois buscar e ler outro, é maior que o custo de buscar (e depois ler) dois registros sucessivos de uma só vez
- Pode-se melhorar o desempenho da busca sequencial **lendo um bloco de registros por vez**, e então processar este bloco em RAM

# Exemplo de blocagem



- Um arquivo com **4.000 registros** cujo tamanho médio é **512 bytes** cada
- A busca sequencial por um registro, sem blocagem, requer em média 2.000 leituras
- Trabalhando com **blocos de 16 registros**, o número médio de leituras necessárias cai para **125** ( dado que há 250 blocos –  $4000/16$ )
- Cada READ gasta um pouco mais de tempo, mas o ganho é considerável devido à redução do número de READs (ou seja, de seeks)

# Blocagem de registros



- Melhora o desempenho, mas o custo continua diretamente proporcional ao tamanho do arquivo, i.e., é  $O(n)$
- Reflete a diferença entre o custo de acesso à RAM e o custo de acesso a disco
  - Aumenta a quantidade de dados transferidos entre o disco e RAM
- Não altera o número de comparações em RAM
- Economiza tempo porque reduz o número de operações seek



## ● Atenção

- Agrupam-se bytes em campos, campos em registros e, agora, registros em blocos
  - Os níveis de organização hierárquica vão aumentando
- Entretanto, **agrupar registros em blocos aumenta o desempenho apenas**, enquanto os demais agrupamentos se relacionam à organização lógica da informação

# Vantagens da Busca Sequencial



- Fácil de programar
- Requer estruturas de arquivos simples

# Busca Sequencial é Razoável



- Na busca por uma cadeia em um arquivo ASCII (como o grep do Unix)
- Em arquivos com poucos registros (da ordem de 10)
- Em arquivos pouco pesquisados (mantidos em fitas, por exemplo)
- Na busca por registros com um certo valor de chave secundária, para a qual se espera muitos registros (muitas ocorrências)



- A alternativa mais radical ao acesso sequencial é o **acesso direto**
- O acesso direto implica em realizar um *seeking* direto para o **início do registro** desejado (ou do setor que o contém) e ler o registro imediatamente
- É **O(1)**, pois um único acesso traz o registro, independentemente do tamanho do arquivo

# Posição do início do registro



## ● Como localizar o início do registro no arquivo

- Para localizar a posição exata do início do registro no arquivo, pode-se utilizar um arquivo de **índice** separado
- Ou se pode ter um **RRN (relative record number) (ou byte offset)** que fornece a posição relativa do registro dentro do arquivo

# Posição de um registro com RRN



- Para utilizar o **RRN**, é necessário trabalhar com registros de tamanho fixo
  - Nesse caso, a posição de início do registro é calculada facilmente a partir do seu RRN
    - *Byte offset = RRN \* Tamanho do registro*
    - Por exemplo, se queremos a posição do registro com RRN 546, e o tamanho de cada registro é 128, o byte offset é **546 x 128 = 69.88**



## ○ Organização a arquivos

- Registros de tamanho fixo
- Registros de tamanho variável

## ○ Acesso a arquivos

- Acesso sequencial
- Acesso direto



- Considerações a respeito da organização do arquivo

- arquivo pode ser dividido em campos?
- os campos são agrupados em registros?
- registros têm tamanho fixo ou variável?
- como separar os registros?
- como identificar o espaço utilizado e o "lixo"?

- Existem muitas respostas para estas questões

- a escolha de uma organização em particular depende, entre outras coisas, do que se vai fazer com o arquivo



- Arquivos que devem conter registros com tamanhos muito diferentes, devem utilizar **registros de tamanho variável**
  - Como acessar esses registros diretamente?
- Existem também **limitações da linguagem**
  - C permite acesso a qualquer byte, e o programador pode implementar acesso direto a registros de tamanho variável
  - Pascal exige que o arquivo tenha todos os elementos do mesmo tipo e tamanho, de maneira que acesso direto a registros de tamanho variável é difícil de ser implementado



- As aplicações usualmente armazenam as informações em arquivos organizando-as em campos e registros. Explique as diferentes maneiras pelas quais um campo pode ser armazenado em um arquivo para posterior recuperação.
- Explique as diferentes estratégias que podem ser utilizadas para separar um registro de outro. Discuta as vantagens e desvantagens de cada uma delas.
- Explique o que é fragmentação de campos e registros. Quando e por que ela ocorre?
- Se a separação entre registros e campos é feita por delimitadores, quais as restrições para a escolha desses delimitadores. Descreva uma situação que exemplifique sua resposta.



- Explique como é possível melhorar o desempenho de um acesso seqüencial a todo o conteúdo de um arquivo. Tal solução também garante um melhor desempenho de uma seqüência arbitrária de acessos aleatórios? Discuta.
- Quais as vantagens e as desvantagens de utilizar arquivos organizados em registros de tamanho fixo?
- O que é RRN? Como é possível fazer acessos aleatórios em arquivos a registros de tamanho fixo?
- É vantajoso manter um arquivo separado para armazenar apenas as chaves e os byte *offsets*, ou RRNs, dos registros no arquivo de dados? Como isto afeta a inserção de um novo registro?



- Você foi contratado para automatizar o cadastro de alunos da UNESP e sua gravação e recuperação de arquivos
- Inicialmente, você pega uma turma de 5 alunos para testar seu programa
- É dado a você o registro abaixo com os seguintes campos:

```
struct aluno {  
    char *nome;  
    char *curso  
    char *ano  
    Char *email  
    int nro_UNESP;  
}
```
- Pede-se: escreva um programa em C que leia na tela os dados; grave os dados da turma em arquivo e recupere o nome de um aluno cujo número UNESP é dado na tela



A função `fseek` reposiciona o indicador de **posição** em um arquivo.

```
1 int fseek(FILE *file, long int offset, int whence);
```

- A função retorna 0 em caso de sucesso e outro valor caso contrário.

Note que a função tem 3 parâmetros:

- ① `file`: ponteiro para o arquivo considerado;
- ② `offset`: quantidade de *bytes* de deslocamento (podemos utilizar números negativos);
- ③ `whence`: indica de onde o deslocamento é feito:
  - `SEEK_SET`: início do arquivo;
  - `SEEK_CUR`: posição atual no arquivo;
  - `SEEK_END`: final do arquivo.



A função **ftell** retorna a **posição** atual em um arquivo (em bytes):

```
1 long int ftell(FILE *file);
```

- A função retorna a **posição** atual no arquivo (em *bytes*).



- FOLK, M.J. File Structures, Addison-Wesley, 1992.
- File Structures: Theory and Practice”, P. E. Livadas, Prentice-Hall, 1990;
- Contém material extraído e adaptado das notas de aula dos professores Moacir Ponti, Thiago Pardo, Leandro Cintra e Maria Cristina de Oliveira.