



# Organização de Arquivos

---

SCE-183 – Algoritmos e Estruturas de Dados II



# Arquivos

---

- Ao construir uma estrutura de arquivos, estamos impondo uma **organização aos dados**
- Qual a diferença entre os termos **stream** e **arquivo**?



# Exercício

---

- Faça um programa em C que
  1. Leia do usuário os seguintes dados de 10 pessoas: nome, idade, número de filhos
  2. Escreva em um arquivo os dados lidos
  3. 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);
}

```

```

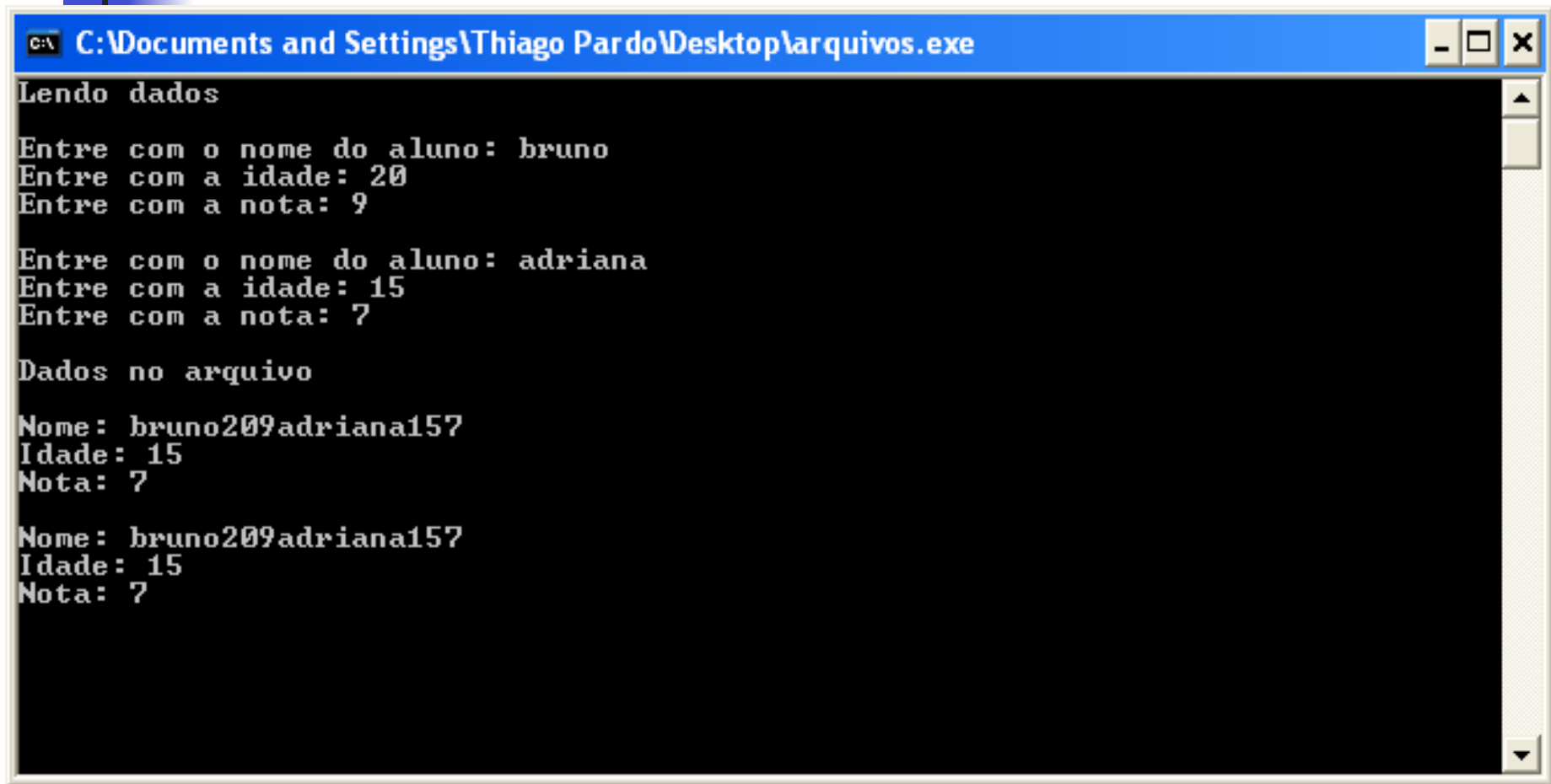
printf("Dados no arquivo\n\n");
f=fopen("saida.txt","r");
for (i=0; i<TAM; i++) {
    fscanf(f,"%s",a.nome);
    printf("Nome: %s\n",a.nome);
    fscanf(f,"%d",&a.idade);
    printf("Idade: %d\n",a.idade);
    fscanf(f,"%d",&a.nota);
    printf("Nota: %d\n\n",a.nota);
}
fclose(f);

return(0);
}

```

O que esse programa vai imprimir?

# Exemplo de execução



```
C:\Documents and Settings\Thiago Pardo\Desktop\arquivos.exe
Lendo dados
Entre com o nome do aluno: bruno
Entre com a idade: 20
Entre com a nota: 9

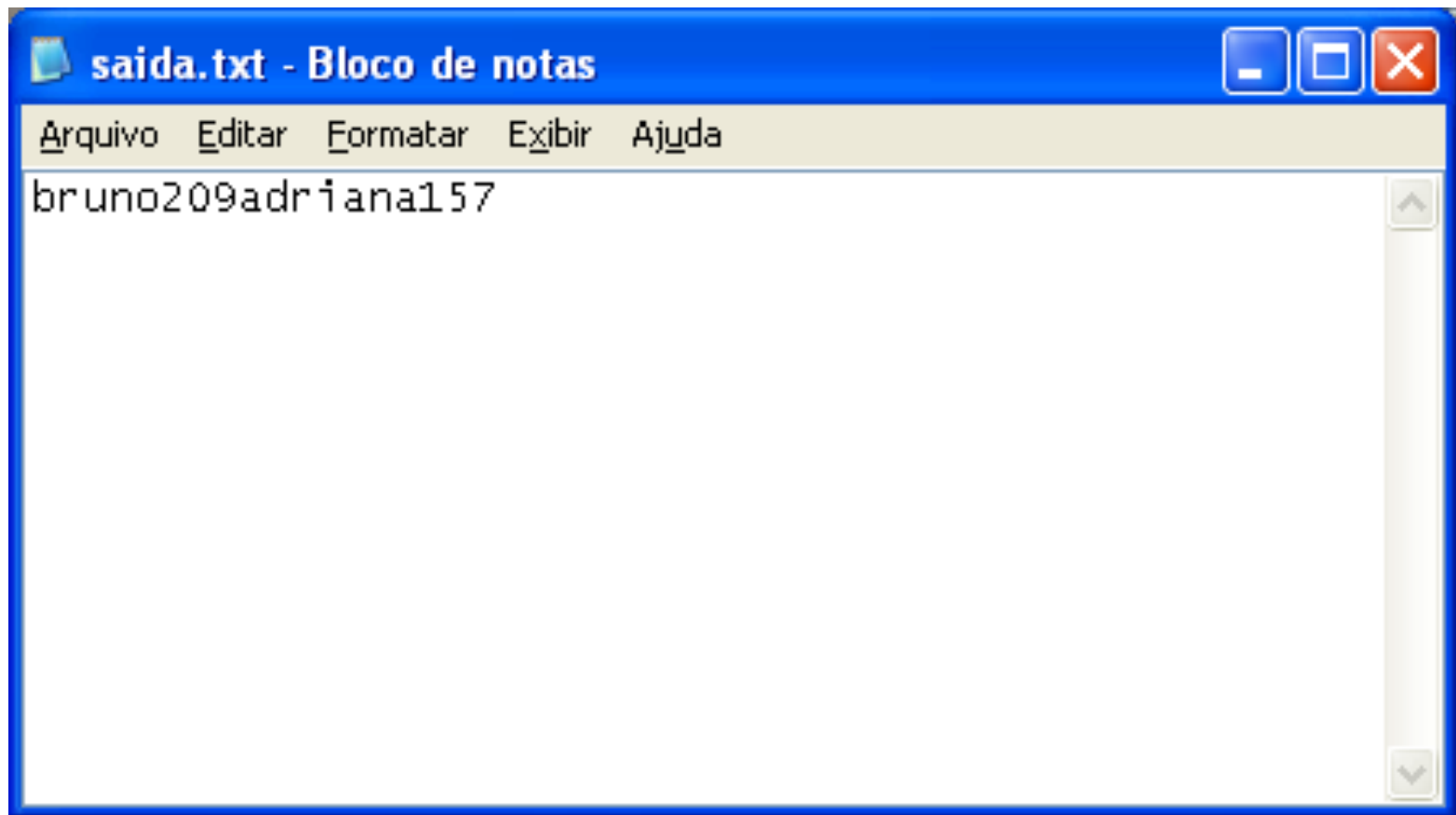
Entre com o nome do aluno: adriana
Entre com a idade: 15
Entre com a nota: 7

Dados no arquivo

Nome: bruno209adriana157
Idade: 15
Nota: 7

Nome: bruno209adriana157
Idade: 15
Nota: 7
```

# Exemplo de execução



The image shows a screenshot of a Windows Notepad window titled "saida.txt - Bloco de notas". The window has a blue title bar with standard minimize, maximize, and close buttons. Below the title bar is a menu bar with the following items: "Arquivo", "Editar", "Formatar", "Exibir", and "Ajuda". The main text area contains a single line of text: "bruno209adriana157". A vertical scrollbar is visible on the right side of the text area.

```
bruno209adriana157
```



# Organização de Arquivos

---

- Informações em arquivos são, em geral, organizadas em **campos** e **registros**
  - **Conceitos lógicos**
    - Não necessariamente correspondem a uma organização física



# Organização de Arquivos

---

- 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 seqüência de bytes (sem delimitadores, contadores, etc.)

AmesJohn123 MapleStillwaterOK74075MasonAlan90 EastgateAdaOK74820





# Organização de Arquivos

---

- 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



# Campos com tamanho fixo

---

```
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?



# Campos com tamanho fixo

---

- 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?

```
05Maria05Rua 10312310São Carlos
04João05Rua A0325509Rio Claro
05Pedro06Rua 10025610Rib. 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 uma *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|
```



## Uso de uma *tag* do tipo "keyword=value"

---

- Outras tecnologias que utilizam esta estratégia?



# 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 necessariamente existe 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|
```

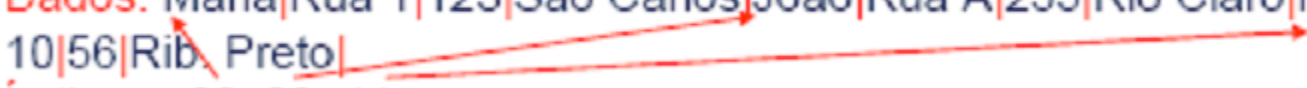


# Utilizar um índice

- 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|Rib. Preto|  
**Índice:** 00 29 44





# Utilizar delimitadores

---

- 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|
```



# fread e fwrite

---

- Vimos que estes comandos escrevem e lêem **registros inteiros** diretamente
  - Por que não usá-los?



# Acesso a registros

---



# 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



# Chaves

---

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



# Chaves Primária e Secundária

---

- Uma **chave primária** é, por definição, a chave utilizada para identificar unicamente um registro
  - Exemplo: número USP, 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



# Chaves Distintas

---

- 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



# Desempenho da Busca

---

- 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**



# Desempenho de Busca

---

- Assumimos que
  - Cada chamada a READ lê 1 registro e requer um *seek*
  - Todas as chamadas a READ tem o mesmo custo



# Busca seqüencial

---

- 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



# Busca seqüencial

---

- 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
    - $1.000/2$ , em média (supondo igual probabilidade de busca por qualquer registro)
  - No pior caso, o trabalho necessário para buscar um registro em um arquivo de tamanho  $n$  utilizando busca seqüencial 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 seqüencial **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 seqüencial 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)
- 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***



# Blocagem de registros

---

- 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 Seqüencial

---

- Fácil de programar
- Requer estruturas de arquivos simples



# Busca seqüencial 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)



# Acesso Direto

---

- A alternativa mais radical ao acesso seqüencial 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* =  $RRN * \text{Tamanho do registro}$
    - Por exemplo, se queremos a posição do registro com RRN 546, e o tamanho de cada registro é 128, *o byte offset é*  $546 \times 128 = 69.888$



# Acesso a arquivos X Organização de arquivos

---

- **Organização de Arquivos**
  - registros de tamanho fixo
  - registros de tamanho variável
  
- **Acesso a arquivos**
  - acesso seqüencial
  - acesso direto





# Acesso a arquivos X Organização de arquivos

---

- 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



# Acesso a arquivos X

## Organização de arquivos

---

- 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



# Exercício

---

- Você foi contratado para automatizar o cadastro de alunos da USP e sua gravação e recuperação de arquivos
- Inicialmente, você pega uma turma de 30 alunos para testar seu programa
- É dado a você o registro abaixo com um campo de tamanho variável:

```
struct aluno {  
    char *nome;  
    int nro_USP;  
}
```

- Pede-se: escreva um programa em C que leia e grave os dados da turma em arquivo e recupere o nome de um aluno cujo número USP é dado utilizando acesso direto ao registro