

# Controle de Registros

*Projeto parte 1*

**Marcos Silvano Almeida**  
Departamento de Computação  
UTFPR Campo Mourão

# Projeto parte 1: Controle de Registros

- Objetivo:
  - Implementar app gerenciamento de registros de dados em memória, simulando um sistema de armazenamento simples
- Características
  - Trabalho em dupla OU sozinho.
  - Interface do usuário (UI) em modo texto com menus de navegação
  - Escolher um tema para os dados: filmes, livros, séries, jogos, esportes, etc
  - Operações: criar, editar, obter, listar e remover registro + ferramentas
    - Confirmação em cada operação
  - Dados são organizados em registros (“string montada por strings”) de tamanho fixo, armazenados sequencialmente como texto em uma única string longa.
    - Implementar várias funcionalidades de manipulação de strings e vetores
  - Cada registro será composto por um número fixo de campos. Exemplo:
    - **REGISTRO DE FILME:** título + ano + diretor

# Projeto parte 1: Controle de Registros

- Considerando o exemplo anterior:

REGISTRO DE FILME

título // 30 letras + \0

ano // 4 números + \0

diretor // 20 letras + \0

Tamanho total do registro: 31 + 5 + 21 = 57 chars

```
char registro[57]; // titulo[31] + ano[5] + diretor[21]
```

- Exemplo de registro de filme (posições vazias são preenchidas com '\0')

“Big Trouble Little China\0\0\0\0\0\0\01986\0John Carpenter\0\0\0\0\0\0\0”

Para simplificar a visualização, representaremos '\0' com 0 nos slides:

“Big Trouble Little China000000019860John Carpenter0000000”

# Estrutura Geral

O vetor **index** funciona como uma tabela. A posição é o ID do registro. O conteúdo, indica se o registro em **data** está utilizado ou livre

	0	1	2	3	4	5	6	7	8	9	⇐ ID do registro
int index[10]	1	1	1	1	1	0	0	0	0	0	⇐ indica vazio (0), utilizado (1) ou removido (-1)

O vetor de chars **data** está representado em linhas para facilitar a visualização. Na prática, os registros ficam um após o outro.

```
char data[570]
```

$$10 \times 57 = 570$$

Tamanho de **data**:  
tamanho do registro  
X  
tamanho de **index**

[illegible]

# Estrutura Geral

- O programa utiliza dois vetores:
  - **char data[n x tamanho\_registro]**: vetor de caracteres para armazenar registros de texto. Cada posição pode armazenar 1 char.
    - Cada registro é formado por uma sequência de caracteres de tamanho fixo
    - Cada registro possui um número fixo de campos
  - **int index[n]**: vetor que agirá como tabela de registros, mapeamento posições livres (0) e ocupadas (1) em **data**.
    - 0 - posição livre: nunca utilizada anteriormente
    - 1 - posição ocupada: existe um registro nesta posição
    - -1 - registro na lixeira: um registro foi removido nesta posição.  
Poderá ser restaurado caso não tenha sido sobrescrito.

**OBS:** Durante a adição de um novo registro, o programa deve tentar encontrar posições livres nunca usadas. Caso não haja, utilizará a primeira posição marcada com -1 (lixeira).

Protótipo de Interface  
+  
Simulação de Funcionamento

# Menu Principal

=====

GERENCIAMENTO DE FILMES

=====

## MENU PRINCIPAL

- |                      |                         |
|----------------------|-------------------------|
| 1 - Adicionar filme  | 5 - Lixeira & Restaurar |
| 2 - Consultar filmes | 6 - Mapa de registros   |
| 3 - Remover filme    | 7 - Desfragmentar Dados |
| 4 - Alterar filme    | 0 - Sair                |

Entre com uma opção: 1

Validar opção (garantir  
que é válida)

# 1 - Adicionar filme

=====

MENU PRINCIPAL / Adicionar filme

=====

Novo ID: [1]

Título: **Batman**

Ano: **1989**

Diretor: **Tim Burton**

Confirmar (S/N)? **S**

Novo registro adicionado.

Pressione uma tecla >

ID é automático (índice do vetor **index**):  
Nesta caso, mostra a próxima posição  
disponível no vetor **index**.

Confirma operação.

Procura a próxima  
posição disponível em  
**index**.

	0	1	2	3	4	5	6	7	8	9
index	1	0	0	0	0	0	0	0	0	0



# 1 - Adicionar filme: como funciona

Procura a próxima posição disponível em **index**.

[illegible]

Concatenar as strings dos campos em um único vetor de chars de 57 posições

Novo registro a ser armazenado:

[illegible]

Copia o vetor de char **registro** para a posição correta em **data**

data

[illegible]

## 2 - Consultar filme

```
=====
MENU PRINCIPAL / Consultar filme
=====
```

```
Busca: man
```

```
Filmes encontrados com 'man':
```

ID	Título	Ano	Diretor
01	Batman	1989	Tim Burton
03	Watchman	2009	Zack Snyder

```
Pressione uma tecla >
```

A busca poderá ser aplicada ao registro inteiro, visto que ele é um vetor de chars.

## 2 - Consultar todos os filmes

Para listar todos os registros, defina algum caractere especial, como asterisco \*. Uma alternativa, é adicionar a opção "Listar Todos" ao menu principal.

=====

MENU PRINCIPAL / Consultar Filme

=====

Busca: \*

Listando todos os registros:

ID	Título	Ano	Diretor
00	They Live	1988	John Carpenter
01	Batman	1989	Tim Burton
02	Aliens	1986	James Cameron
03	Watchman	2009	Zack Snyder
04	Big Trouble Little China	1986	John Carpenter

Pressione uma tecla >

### 3 - Remover filme

O processo para remover um filme existente utiliza o processo de:

- Consultar filme

MENU PRINCIPAL / Editorar filme

Busca: **alien**

Filmes encontrados com 'alien':

ID	Título	Ano	Diretor
02	Aliens	1986	James Cameron

A posição removida em **index** é marcada como “lixeira” (-1). Em **data**, os dados do registro são mantidos, caso seja necessário restaurá-lo.

ID a remover: 2

index	1	1	-1	1	1	0	0	0	0	0
-------	---	---	----	---	---	---	---	---	---	---

Confirmar (S/N) ? **S**

Registro na lixeira.

data

[illegible]

## 4 - Alterar filme

=====

MENU PRINCIPAL / Editar filme

=====

Busca: **man**

Filmes encontrados com 'man':

ID	Título	Ano	Diretor
01	Batman	1989	Tim Burton
03	Watchman	2009	Zack Snyder

Informe o ID: 1

Título anterior: <b>Batman</b>	=>	Novo: <b>Batman 2</b>
Ano anterior: <b>1989</b>	=>	Novo: <b>1992</b>
Diretor anterior: <b>Tim Burton</b>	=>	Novo: <b>Tim Burton</b>

Confirmar (S/N)? **S**

Registro anterior removido. Novo registro adicionado.

O processo para alterar um filme existente utiliza os processo de:

- Consultar filme
- Remover filme (antigo)
- Adicionar filme (novo)

## 5 - Lixeira & Restaurar

MENU PRINCIPAL / Lixeira & Restaurar

Filmes encontrados na lixeira:

ID	Título	Ano	Diretor
02	Aliens	1986	James Cameron
03	Watchmen	1986	Zack Snyder

Exibe os registros marcados com -1.  
Caso **restaurado**, retorna para 1.

Selezione o ID a restaurar: 2

## index

1	1	-1	-1	1	0	0	0	0	0
---	---	----	----	---	---	---	---	---	---

Confirmar (S/N) ? **S**

Registro restaurado.

They Live0000000000000000000000000000000019880John Carpenter000000  
Batman0000000000000000000000000000000019890Tim Burton000000000000  
Aliens0000000000000000000000000000000019860John Cameron0000000000  
Watchmen0000000000000000000000000000000020090Zack Snyder0000000000  
Big Trouble Little China000000019860John Carpenter0000000  
00  
00  
00  
00  
00

data

## 6 - Mapa de Registros

MENU PRINCIPAL / Mapa de Registros

Exibir as duas estruturas de dados do programa: **index** e **data**

Vetor INDEX: 4 / 10 índices

$$[1, 1, -1, 1, 1, 0, 0, 0, 0, 0]$$

Vetor DATA: 570 caracteres

[illegible][illegible][illegible][illegible]

Big Trouble Little China000000019860John Carpenter0000000

[illegible][illegible][illegible][illegible][illegible]

## 7 - Desfragmentar Dados

Desfragmentar:  
Mover as posições ocupadas para a esquerda, sobre as posições livres (em **index** e **data**).

MENU PRINCIPAL / Desfragmentar Dados

Vetor INDEX: 4 / 10 índices

$$[1, 1, -1, 1, 0, 0, 0, 1, 0, 0]$$

Vamos considerar um armazenamento com mais “buracos”

Vetor DATA: 570 caracteres

[illegible][illegible][illegible][illegible][illegible][illegible][illegible]

Big Trouble Little China000000019860John Carpenter0000000

[illegible][illegible]



## 7 - Desfragmentar Dados (continuação)

Confirmar (S/N)? **S**

Desfragmentação em progresso... concluída.

Vetor INDEX: 4 / 10 índices

[1, 1, 1, 1, 0, 0, 0, 0, 0, 0]

Todas as posições ocupadas foram movidas para a esquerda em **index** e **data**. As posições livres estão agora todas à direita.

Vetor DATA: 570 caracteres

They Live0000000000000000000000000000000019880John Carpenter00000000  
Batman0000000000000000000000000000000019890Tim Burton000000000000  
Watchman0000000000000000000000000000000020090Zack Snyder0000000000  
Big Trouble Little China0000000019860John Carpenter00000000  
00  
00  
00  
00  
00  
00  
00

## Trabalhando com Strings

# Operações com string

- Vários exercícios e exemplos são importantes para a realização do projeto:
  - `int strlen(char str[]):` calcula tamanho da string
  - `int strcmp(char str1[], char str2[]):` verifica igualdade de strings
  - `int strstr(char dest[], char src[]):` verifica se encontra src está em dest
  - `void strToUpper(char str[]):` converte todas as letras para maiúsculas
  - `void strTrim(char str[]):` remove espaços antes e depois da string
  - `void strcat(char dest[], char src[]):` acrescenta string src ao final de dest
  - Outras...
- OBS: não é permitido utilizar a biblioteca **string.h** ou outra biblioteca que efetue operações sobre strings (exceto `printf()`).

# Lendo strings com **scanf()**

- Existem várias funções na biblioteca de C padrão que podem ser usadas para leitura de strings
  - `scanf()`, `getline()`, `fgets()` e `gets()` (**depreciada**)
  - **Scanf()** da `<stdio.h>` é bastante versátil, pois permite expressões de leitura.
    - Podemos filtrar quais caracteres devem ser lidos
    - Podemos limitar a quantidade caracteres a serem lidos
- Alguns problemas ao ler texto (strings):
  - Digitar um texto mais longo que a string que o receberá
  - Por padrão, **scanf** termina ao encontrar o primeiro espaço ou quebra de linha (`\n`)
    - O restante (incluindo o `\n`), fica no buffer de entrada do teclado e será lido automaticamente no próximo `scanf()`, inutilizando os próximos `scanf()`
- Solução:
  - Usar expressão que limita a quantidade de caracteres lidos no `scanf()`
  - Limpar o buffer de entrada após cada `scanf()`

# Lendo strings com **scanf()**

- Solução: Usar expressão que limita a quantidade de caracteres lidos e sempre limpar o buffer de entrada após cada scanf().

```
void clearBuffer() {  
    scanf("%*[^\\n]"); // faz a leitura de vários caracteres (*) e encerra no \\n ([^\\n])  
    scanf("%*c");      // lê o próximo caractere: neste caso, o '\\n'  
}
```

...

```
char s[10];  
scanf(" %9[^\\n]", s); // lê 9 caracteres do teclado e concatena \\0 ao final  
clearBuffer();        // limpa o restante do buffer de entrada, se houver
```

```
int num;  
scanf(" %d", &num);  
clearBuffer();        // limpa o restante do buffer de entrada, se houver
```

Trabalhando com cores no terminal

# Biblioteca “screen.h”

- Biblioteca “**screen.h**” permitirá criar uma interface mais interessante
- Para utilizar a biblioteca deve-se:
  - Incluir os arquivos na pasta de seu projeto: **screen.h** + **screen.c**
  - Incluir a biblioteca em seu código: **#include “screen.h”**

A definição de cores funciona na maioria dos terminais para **Linux**, além de terminais web, como **repl.it**

```
void main() {  
    setColor(COLOR_FG_BLUE); // cor AZUL para caracteres  
    printf("\nUSANDO CORES NO TERMINAL\n");  
  
    setColor(COLOR_BG_WHITE); // cor BRANCO para fundo  
    setColor(COLOR_FG_BLACK); // cor PRETO para caracteres  
    printf("OBS: nem todos os terminais oferecem suporte\n\n");  
    resetColor();  
}
```

# Biblioteca “screen.h”

- Funções importantes da biblioteca screen.h

```
// define a posição do cursos na tela
void setCursor(int row, int col);
// restaura configuração de cores do terminal
void resetColor();
// define cor no terminal
//     FG = foreground = cor do caractere
//     BG = background = cor do fundo do caractere
void setColor(int color);
// limpa a tela com a cor de fundo atual
void clearScreen();
// sorteira cor para fundo (BG) ou caractere (FG)
void randomBGColor();
void randomFGColor();
```



## Avaliação do Projeto

# Critérios de Avaliação do Projeto

- Qualidade/eficiência das soluções utilizadas
  - Algoritmos implementados
- Organização do código
  - Legibilidade do código (identação, identificadores, ...)
  - Documentação do código (comentários)
- Estrutura do código
  - Modularização em funções para organização e evitar duplicações
- Acabamento da interface em texto
- Divisão das atividades na dupla
  - Divisão balanceada
- Conhecimento sobre o programa implementado
  - Apresentação síncrona com questionamentos do professor
  - Cada aluno é responsável por sua parte

# Questões sobre Avaliação

- Cópias: Qualquer tipo de cópia (trabalhos de colegas, internet, etc) anulará imediatamente o trabalho
  - Seja por porções de código ou pelo trabalho completo
  - Projeto deve ser de autoria exclusiva dos integrantes da equipe
  - Cada aluno deve necessariamente conhecer o seu código
    - Estruturas utilizadas
    - Algoritmos implementados
- Entrega:
  - ZIP com código fonte pelo Moodle
  - Apresentação síncrona + Responder questionamentos do professor