

CCP130

Desenvolvimento de Algoritmos

Prof. Danilo H. Perico



Registros

Registros



- De forma geral, **registros** referem-se ao armazenamento de dados
- **Registros são coleções de variáveis agrupadas, referenciadas por um nome**
- Registros são geralmente **heterogêneos** (tipos diferentes de dados)
- Cada elemento do registro é chamado de ***campo***
- Registros são também conhecidos como:
 - ***estrutura*** ou
 - ***struct***

Exemplo - Registro de Empregado

REGISTRO DE EMPREGADO											
EMPREGADOR				ENDEREÇO							
NÚMERO DE ORDEM				NOME						Nº DA MATRÍCULA	
FOTO 3 X 4	FILIAÇÃO		PAI				NACIONALIDADE				
			MÃE				NACIONALIDADE				
	DATA DO NASCIMENTO		IDADE	NACIONALIDADE		ESTADO CIVIL		LOCAL DE NASCIMENTO		ESTADO	CÉDULA DE IDENTIDADE
	CART. PROFISSIONAL		SÉRIE	CART. RESERVISTA		CATEGORIA	CPF / CIC		TÍTULO DE ELEITOR		CART. DE SAÚDE
QUANDO ESTRANGEIRO			CART. MOD. 19		E CASADO COM BRASILEIRA?			E NATURALIZADO?		TEM FILHOS BRASILEIROS?	
DATA QUE CHEGOU AO BRASIL			Nº REG. GERAL		NOME DO CÔNJUGE			QUANTOS?		AUTENTICAÇÃO	
ENDEREÇO						CARACTERÍSTICAS FÍSICAS					
MUDANÇA DE ENDEREÇO						ALTURA	PESO	CABELO	OLHOS	SINAIS	
BENEFICIÁRIOS	NOME			PARENTESCO		NASCIDO EM		PROGRAMA DE INTEGRAÇÃO SOCIAL - (PIS)			
								CADASTRADO EM			
								SOB Nº			
								DEP NO BANCO			
								ENDEREÇO			
								CÓDIGOS			
							BANCO		AGÊNCIA		

Registros

- Em C, os registros são chamados de **struct**
- **structs** são coleções de variáveis agrupadas
- **structs definem tipos que agrupam variáveis**
- As variáveis de uma **struct** podem ter tipos diferentes
- Sintaxe:

```
struct nome_struct {  
    tipo nome_da_variável;  
    tipo nome_da_variável;  
    tipo nome_da_variável;  
};
```

struct - Exemplos

```
struct sPonto {  
    int x;  
    int y;  
};
```

- Esta é uma estrutura que representa um ponto em coordenadas cartesianas
- Guarda duas variáveis inteiras: **x** e **y**

struct - Exemplos

```
struct sPonto {  
    int x;  
    int y;  
};
```

- Por enquanto, apenas a estrutura foi definida, porém não temos uma variável do tipo *struct*.

struct - Exemplos

```
struct sPonto {  
    int x;  
    int y;  
};
```

- Sintaxe para declarar uma variável do tipo *struct*:

```
int main(){  
    struct sPonto meuPonto;  
}
```


struct - Exemplos

- Outra forma declarar uma variável do tipo *struct* é na própria definição da estrutura:

```
struct sPonto {  
    int x;  
    int y;  
};
```



Coloque aqui o nome(s) de sua
variável(eis)

struct - Exemplos

- Outra forma declarar uma variável do tipo *struct* é na própria definição da estrutura:

```
struct sPonto {  
    int x;  
    int y;  
} meuPonto;
```



Coloque aqui o nome(s) de sua
variável(eis)

Exemplo - *struct* de pessoa

- Sintaxe para criação de um registro que armazena dados de pessoa:

```
struct pessoa {  
    char nome[50];  
    int idade;  
    char sexo;  
};
```

Declarando algumas pessoas

- A partir da mesma **struct**, podemos criar várias variáveis. Exemplo:

```
int main(void){  
    struct pessoa p1;  
    struct pessoa p2;  
    struct pessoa p3;  
    return 0;  
}
```

Acessando os elementos de uma variável do tipo *struct*

- Para acessar qualquer elemento de uma estrutura, basta colocar o identificador da variável estrutura seguido do nome do elemento. Exemplo:

```
strcpy(p1.nome, "Fulano");  
strcpy(p2.nome, "Sicrana");  
strcpy(p3.nome, "Beltrano");
```

```
p1.idade = 26;  
p2.idade = 30;  
p3.idade = 18;
```

```
p1.sexo = 'M';  
p2.sexo = 'F';  
p3.sexo = 'M';
```

Acessando os elementos de uma variável do tipo *struct*


strcpy - copy string

Depois de inicializados não podemos alterar o valor de um vetor inteiro

strcpy atribui a string para o vetor de char

Precisa da biblioteca:

#include <string.h>



```
strcpy(p1.nome, "Fulano");  
strcpy(p2.nome, "Sicrana");  
strcpy(p3.nome, "Beltrano");
```

```
p1.idade = 26;  
p2.idade = 30;  
p3.idade = 18;
```

```
p1.sexo = 'M';  
p2.sexo = 'F';  
p3.sexo = 'M';
```

Acessando os elementos de uma variável do tipo *struct*

- Para acessar qualquer elemento de uma estrutura, basta colocar o identificador da variável estrutura seguido do nome do elemento. Exemplo:

```
printf("Nome de p1: %s\n", p1.nome);  
printf("Nome de p2: %s\n", p2.nome);  
printf("Nome de p3: %s\n", p3.nome);
```

Vetor de registros

- Se eu precisar de várias pessoas, vou ter que declarar uma por uma?

```
int main(void){  
    struct pessoa p1;  
    struct pessoa p2;  
    struct pessoa p3;  
    return 0;  
}
```

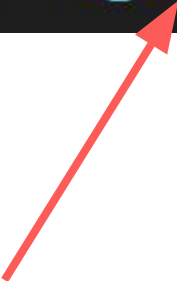

Vetor de registros

Não!

Vetor de registros

- Podemos fazer um vetor de registros:

```
struct pessoa agenda[100];
```



Chamei de “agenda” meu vetor de pessoas e declarei o tamanho como 100 - ou seja - terei 100 pessoas na agenda

Vetor de registros

- Com o vetor os acessos aos elementos ficam assim
(estou fazendo somente para os 3 primeiros):

```
strcpy(agenda[0].nome, "Fulano");  
strcpy(agenda[1].nome, "Sicrana");  
strcpy(agenda[2].nome, "Beltrano");
```

```
agenda[0].idade = 26;  
agenda[1].idade = 30;  
agenda[2].idade = 18;
```

```
agenda[0].sexo = 'M';  
agenda[1].sexo = 'F';  
agenda[2].sexo = 'M';
```

```
printf("Nome da 1a pessoa: %s\n", agenda[0].nome);  
printf("Nome de 2a pessoa: %s\n", agenda[1].nome);  
printf("Nome de 3a pessoa: %s\n", agenda[2].nome);
```

```
4 struct pessoa {
5     char nome[50];
6     int idade;
7     char sexo;
8 };
9
10 void imprime(struct pessoa p){
11     printf("%s\n", p.nome);
12     printf("%d\n", p.idade);
13     printf("%c\n", p.sexo);
14 }
15
16 int main(void){
17     struct pessoa agenda[100];
18     strcpy(agenda[0].nome, "Fulano");
19     agenda[0].idade = 26;
20     agenda[0].sexo = 'M';
21     imprime(agenda[0]);
22     return 0;
23 }
```

structs e funções

- Podemos passar estruturas inteiras como **parâmetros de funções**

Função imprime tem 1 parâmetro: uma struct pessoa.

Chamada da função imprime - somente uma pessoa é enviada como argumento

Registros e Ponteiros

- Cada *struct* tem um endereço na memória do computador
- É muito comum usar um ponteiro para armazenar o endereço de uma *struct*
- Dizemos que um tal ponteiro aponta para o registro

structs e funções

- Podemos passar estruturas inteiras por **referência para funções**

Função `leiaDados` tem 1 parâmetro: um ponteiro de struct `pessoa`.

Chamada da função - o endereço de uma pessoa é enviado

```
3 struct pessoa {
4     char nome[50];
5     int idade;
6     char sexo;
7 };
8
9 void leiaDados(struct pessoa *p){
10     printf("Digite o nome: ");
11     scanf("%s", (*p).nome);
12     printf("Digite a idade: ");
13     scanf("%d", &(*p).idade);
14     printf("Digite o sexo: ");
15     scanf(" %c", &(*p).sexo);
16 }
17
18 int main(void){
19     struct pessoa agenda[100];
20     leiaDados(&agenda[0]);
21     printf("Dados: %s %d %c", agenda[0].nome, agenda[0].idade, agenda[0].sexo);
22     return 0;
23 }
```

structs e funções

```
3 struct pessoa {
4     char nome[50];
5     int idade;
6     char sexo;
7 };
8
9 void leiaDados(struct pessoa *p){
10     printf("Digite o nome: ");
11     scanf("%s", (*p).nome);
12     printf("Digite a idade: ");
13     scanf("%d", &(*p).idade);
14     printf("Digite o sexo: ");
15     scanf(" %c", &(*p).sexo);
16 }
17
18 int main(void){
19     struct pessoa agenda[100];
20     leiaDados(&agenda[0]);
21     printf("Dados: %s %d %c", agenda[0].nome, agenda[0].idade, agenda[0].sexo);
22     return 0;
23 }
```

Podemos escrever estas linhas de um jeito melhor!

structs e funções

```
3 struct pessoa {
4     char nome[50];
5     int idade;
6     char sexo;
7 };
8
9 void leiaDados(struct pessoa *p){
10     printf("Digite o nome: ");
11     scanf("%s", p->nome);
12     printf("Digite a idade: ");
13     scanf("%d", &p->idade);
14     printf("Digite o sexo: ");
15     scanf(" %c", &p->sexo);
16 }
17
18 int main(void){
19     struct pessoa agenda[100];
20     leiaDados(&agenda[0]);
21     printf("Dados: %s %d %c", agenda[0].nome, agenda[0].idade, agenda[0].sexo);
22     return 0;
23 }
```

Usando o operador `->`, no lugar de `*` e `&`.

structs

- Alocando dinamicamente

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct sRegistro{
5      int a;
6      int b;
7  };
8
9  void leiaTudo(struct sRegistro *reg){
10     printf("Insira o valor de a: ");
11     scanf("%d", &reg->a);
12     printf("Insira o valor de b: ");
13     scanf("%d", &reg->b);
14 }
15
16 int main(void){
17     struct sRegistro *reg = malloc(sizeof(struct sRegistro));
18     leiaTudo(reg);
19     printf("%d\n", reg->a);    // utilizando ->
20     printf("%d\n", (*reg).b); // ou utilizando * e .
21     free(reg);
22 }
```

Exemplo



Exemplo

Declare uma **struct** para armazenar o nome, idade, código, sexo e salário de um **Funcionário**.

Construa uma função responsável por declarar um Funcionário e por receber os dados deste Funcionário do usuário.

Escreva uma função para exibir os dados do Funcionário declarado na função anterior. A função deve receber um parâmetro do tipo Funcionário.

Exercícios

Exercícios



1. Defina uma **struct** que irá representar bandas de música. Essa estrutura deve ter o nome da banda, que tipo de música ela toca e o número de integrantes.
2. Crie um procedimento para preencher 5 estruturas de bandas criadas no exercício anterior. Após criar e preencher, exiba todas as informações das bandas/estruturas. Não se esqueça de usar o operador **->** para preencher os membros das structs.