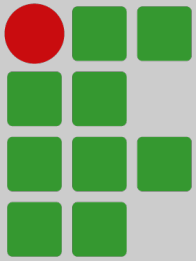

INSTITUTO FEDERAL

Campus Charqueadas

Programação

Structs

Prof. Silvana Teodoro

Struct

- São variáveis compostas heterogêneas.
- São conjuntos de dados logicamente relacionados, mas de tipos diferentes (inteiro, real, string, etc.)
- Os elementos dos registros são chamados de campos.
- Exemplo: Dados sobre funcionários de uma empresa:
  - Nome
  - Idade
  - Salário

Struct

```
struct nome_do_tipo_do_registro {
    tipo1 campo1;
    tipo2 campo2;
    tipo3 campo3;
    // ...
    tipon campon;
};
```

```
struct funcionario {
    char nome[50];
    int idade;
    float salario;
};
```

Struct
Acesso a campos de um registro

- Pode ser realizado através da seguinte sintaxe:

  nome_do_registro.nome_do_campo

- Para uma variável **f** do tipo **funcionario**:

  struct funcionario f;

- O campo nome é acessado assim:

  f.nome

Struct
Exemplo

```c
#include <stdio.h>
#include <string.h>

struct funcionario {
    char nome[50];
    int idade;
    float salario;
};

int main() {
    struct funcionario f;
    strcpy(f.nome,"Regis");
    f.idade = 18;
    f.salario = 1000;
    printf("Nome: %s\n", f.nome);
    printf("Idade: %d\n", f.idade);
    printf("Salario: %.2f\n", f.salario);
    return 0;
}
```

Struct
Vetor de Registros

- Declaração:

```
struct nome_do_registro nome_da_variavel[tamanho_do_vetor];
```

- Uso:

```
nome_da_variavel[indice].nome_do_campo
```

Struct
Vetor de Registros

- **Exemplo**

```c
#include <stdio.h>
#include <string.h>

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

int main() {
  struct pessoa p[2];
  strcpy(p[0].nome,"Regis");
  p[0].idade = 18;
  strcpy(p[1].nome,"Maria");
  p[1].idade = 25;
  printf("Nome: %s - Idade: %d\n", p[0].nome, p[0].idade);
  printf("Nome: %s - Idade: %d\n", p[1].nome, p[1].idade);
  return 0;
}
```

Struct
Funções e registros

- Funções podem aceitar registros como parâmetros e devolver registros como resultado.

- Regra simples:

  struct nome é, para o C, um tipo tal qual int e float, e pode ser usado como os primitivos nas funções.

Struct
Funções e registros

- **Exemplo**

```
#include <stdio.h>
#include <string.h>

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

void imprime_pessoa(struct pessoa p) {
   printf("nome: %s\n", p.nome);
   printf("idade: %i\n", p.idade);
}

int main() {
   struct pessoa aluno1;
 …
   imprime_pessoa(aluno1);
}
```

Struct
Ponteiros

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

- **Exemplo:**
  - data *p; /* p é um ponteiro para registros data */
  - data x;
  - **p = &x**; /* agora p aponta para x */
  - (***p**).dia = 31; /* mesmo efeito que x.dia = 31
    A expressão *p.dia equivale a *(p.dia)) */

- A expressão **p->mes** é uma abreviatura muito útil para a expressão **(*p).mes** :
  - **p->dia** = 31; /* mesmo efeito que (*p).dia = 31 */

Struct
Ponteiros

- **Exemplo**

```
typedef struct
{
    char nome[100];
    int idade;
} pessoa;

main()
{
    pessoa joao;
    pessoa *p = &joao;

    strcpy(joao.nome, "joao da silva");
    joao.idade = 20;
    printf("%s, %d\n", (*p).nome, (*p).idade);

    (*p).idade = 18;
    printf("%s, %d\n", joao.nome, joao.idade);
}
```

Struct
Ponteiros

- **Exemplo**

```
typedef struct
{
    char nome[100];
    int idade;
} pessoa;

main()
{
    pessoa joao;
    pessoa *p = &joao;

    strcpy(joao.nome, "joao da silva");
    joao.idade = 20;
    printf("%s, %d\n", p->nome, p->idade);

    p->idade = 18;
    printf("%s, %d\n", joao.nome, joao.idade);
}
```

Struct
Alocação dinâmica de Memória

- Permite solicitar memória em tempo de execução

- Função para alocar memória (stdlib.h):
  - **malloc(num_bytes)**
  - Retorna o endereço de memória da região alocada
  - Retorna zero se não for possível alocar

- Função para liberar a memória (stdlib.h):
  - **free(endereco_regiao_alocada)**
  - A região fica disponível para outras variáveis/alocações

Struct
Alocação dinâmica de Memória

```c
typedef struct
{
    char nome[100];
    int idade;
} pessoa;

main()
{
    // alocando uma estrutura
    pessoa *p = (pessoa*) malloc(sizeof(pessoa));
    if (p)
    {
        p->idade = 3;
        printf("%d\n", p->idade);
        free(p);
    }
}
```

Struct
Alocação dinâmica de Memória

```
main()
{
     // alocando um vetor com 3 inteiros
     int *v = (int*) malloc( 3 * sizeof(int) );
     if (v)
     {
          v[0] = 10;
          v[1] = 20;
          v[2] = 30;
          printf("%d %d %d\n", v[0], v[1], v[2]);
          free(v);
     }
}
```

Exercícios

Exercícios

1. Criar uma estrutura chamada DadosAluno, que armazena a média e idade de um aluno. Na função main: criar uma variável que é uma estrutura DadosAluno; ler a média e a idade de um aluno e armazenar na variável criada; exibir na tela a média e a idade do aluno.

2. Considerando o exercício 1, criar uma variável que é um vetor da estrutura DadosAluno. O programa deve obter a média e a idade de 10 alunos. Depois, estes dados devem ser exibidos.

Exercícios

4. Crie um programa para armazenar uma agenda de contatos pessoais usando estrutura. Faça uma função para listar contatos. Um contato deve ser composto por nome, endereço, telefone e e-mail.

   Use alocação dinâmica para o armazenamento do nome do contato. Para definir e armazenar a lista de contatos, use um vetor de estrutura, também alocado dinamicamente.