REGISTROS (STRUCT)

Prof. Muriel Mazzetto

Algoritmos 2

Tipos de dados

- Tipos básicos: char, int, float, double.

```
int qtd = 0, i;
char resp = 'n';
int contatos = 100;
int letras = 100, digitos = 10;
```

- Tipos compostos homogêneos: vetores e matrizes.

```
char Nome[contatos][letras + 1];
char RA[contatos][digitos + 1];
```

Problema

- Organizar um cadastro de diferentes informações de diferentes contatos:
  - Armazenar Nome, data de nascimento, RA, patente no CS, etc.

Problema

- Organizar um cadastro de diferentes informações de diferentes contatos:
  - Armazenar Nome, data de nascimento, RA, patente no CS, etc.
- Criar vetores e matrizes para cada dado.
- Localizar pelo índice.
- Manter estruturas avulsas dentro do mesmo código.

Problema

- Org
  - de
    - A
      C
- Cri
- Loc
- Ma

ações

ente no

ódigo.

```
int main(void)
{
    int qtd = 0, i;
    char resp = 'n';

    int contatos = 100;
    int letras = 100, digitos = 10;

    char Nome[contatos][letras + 1];
    char RA[contatos][digitos + 1];
    int dia[contatos], mes[contatos], ano[contatos];

    do
    {
        printf("Deseja inserir um novo contato (s/n)?: ");
        scanf(" %c", &resp);
        if(qtd == contatos)
        {
            printf("Lista cheia.\n");
            resp = 'n';
        }
        else if(resp == 's')
```

Problema


```
        else if(resp == 's')
        {
            printf("Informe o nome: ");
            scanf(" %[^\n]s", Nome[qtd]);
            printf("Informe o RA: ");
            scanf(" %s", RA[qtd]);
            printf("Informe o dia de nascimento: ");
            scanf("%d", &dia[qtd]);
            printf("Informe o mes de nascimento: ");
            scanf("%d", &mes[qtd]);
            printf("Informe o ano de nascimento: ");
            scanf("%d", &ano[qtd]);
            qtd++;
        }

    }while(resp == 's');

    for(i = 0; i < qtd; i++)
    {
        printf("Nome[%d] = %s\n", i, Nome[i]);
        printf("RA[%d] = %s\n", i, RA[i]);
        printf("Nascimento: %d/%d/%d \n\n", dia[i], mes[i], ano[i]);
    }

    return 0;
}
```

Problema

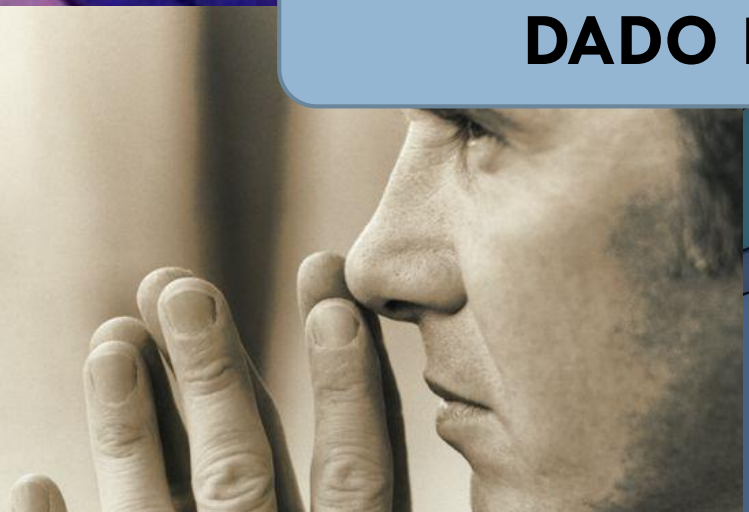

```
        else if(resp == 's')
        {
            printf("Informe o nome: ");
            scanf(" %[^\n]s", Nome[qtd]);
            pr
            s
            p
            s
            p
            s
            p
            s
            q
        }

    }while(re

    for(i = 0;
    {
        printf("Nome[%d] = %s\n", i, Nome[i]);
        printf("RA[%d] = %s\n", i, RA[i]);
        printf("Nascimento: %d/%d/%d \n\n", dia[i], mes[i], ano[i]);
    }

    return 0;
}
```

DIFÍCIL DE ORGANIZAR E MODIFICAR OS DADOS.

DIFICULDADE EM MANTER INTEGRIDADE ENTRE OS DADOS E SEUS ÍNDICES.

Problema

COMO AGRUPAR **DIFERENTES TIPOS DE DADO** PELO **MESMO <u>NOME</u>?**



Registro (Struct)

- Um registro é um "pacote" de variáveis, possivelmente de **tipos diferentes.**

- Cada variável é um **campo** do registro.
  - Na linguagem C, registros são conhecidos como ***struct***.

- São **variáveis heterogêneas,** ou seja, compostas por mais de um tipo de dado diferente.

Registro (Struct)

- Permite criar tipos de dados personalizados.
  - Possui um nome do conjunto de dados (nome do registro).
  - Cada campo possui seu próprio identificador.

```
struct nome_registro{
    tipo1 campo1;
    tipo2 campo2;
    ...
    tipoN campoN;
};
```

Registro (Struct)


```c
#include <stdio.h>
//Definição do registro
struct aluno{
    char nome[81];
    char endereco[121];
    int RA;
};
```

```c
int main(void)
{
    //Declaração de variáveis
    char nome[81];
    char endereco[121];
    int RA;

    //Declaração de registro
    struct aluno a;

    return 0;
}
```

Registro (Struct)


```c
#include <stdio.h>
//Definição do registro
struct aluno{
    char nome[81];
    char endereco[121];
    int RA;
};
```

DEFINIÇÃO GLOBAL:
Todo o código poderá declarar e utilizar variáveis do novo tipo de dado.

```c
int main(void)
{
    //Declaração de variáveis
    char nome[81];
    char endereco[121];
    int RA;

    //Declaração de registro
    struct aluno a;

    return 0;
}
```

Registro (Struct)


```c
#include <stdio.h>
//Definição do registro
struct aluno{
    char nome[81];
    char endereco[121];
    int RA;
};
```

```c
int main(void)
{
    //Declaração de variáveis
    char nome[81];
    char endereco[121];
    int RA;

    //Declaração de registro
    struct aluno a;

    return 0;
}
```

CONJUNTO DE VARIÁVEIS: Agrupa diferentes tipos de dados sob um identificador único.

Operações com Struct


```c
#include <stdio.h>
//Definição do registro
struct aluno{
    char nome[81];
    int RA;
    float coef;
};
```

```c
int main(void)
{
    //Declaração de registro
    struct aluno a;

    /*Cada campo (variável) da struct pode
    ser acessado pelo operador '.' (ponto)*/
    a.RA = 1399999;//!Atribuição
    printf("RA: %d\n", a.RA);//!Impressão
    scanf("%f", &a.coef);//!Leitura
    scanf(" %[^\n]s", a.nome);//!Leitura string
    printf("Aluno: %s", a.nome);//!Impressão string
    a.coef = (a.coef*100) + (a.coef/10);//!Expressão

    return 0;
}
```

Operações com Struct


```c
#include <stdio.h>

struct ponto2D{
    float x, y;
};

struct ponto3D{
    float x, y, z;
};
```

```c
int main(void)
{
    /*Estruturas diferentes podem ter
    campos com o mesmo nome*/

    struct ponto2D p_2D;
    struct ponto3D p_3D;

    p_2D.x = 5.5;
    p_3D.x = 10.10;

    printf("Ponto 2D: %f\n", p_2D.x);
    printf("Ponto 3D: %f\n", p_3D.x);

    return 0;
}
```

Inicialização de Struct


```c
#include <stdio.h>

struct aluno{
    char nome[81];
    int RA;
    float coef;
};

int main(void)
{
    /*Inicialização com uma lista de valores
    seguindo a ordem dos dados da struct*/
    struct aluno a = {"Muriel", 1399999, 0.896};
    printf("A: %s - %d - %f\n", a.nome, a.RA, a.coef);

    return 0;
}
```

Inicialização de Struct


```c
#include <stdio.h>

struct aluno{
    char nome[81];
    int RA;
    float coef;
};

int main(void)
{
    /*Inicialização com uma lista de valores
    seguindo a ordem dos dados da struct*/
    struct aluno a = {"Muriel", 1399999, 0.896};
    printf("A: %s - %d - %f\n", a.nome, a.RA, a.coef);

    return 0;
}
```

Para inicializar é necessário seguir a **ordem dos elementos na definição** da *struct*.

Atribuição entre Structs


```c
struct aluno{
    char nome[81];
    int RA;
    float coef;
};

int main(void)
{
    struct aluno a = {"Muriel", 1399999, 0.896};
    printf("A: %s - %d - %f\n", a.nome, a.RA, a.coef);

    //!É possivel atribuir diretamente uma struct em outra.
    //!Somente se as estruturas forem do mesmo tipo.
    //!Cada valor é copiado para seu respectivo campo.
    struct aluno b;
    b = a;
    printf("B: %s - %d - %f\n", b.nome, b.RA, b.coef);

    return 0;
}
```

Atribuição entre Structs

SOMENTE SE FOREM VARIÁVEIS DA **MESMA ESTRUTURA.**

```c
struct aluno{
    char nome[81];
    int RA;
    float coef;
};

int main(void)
{
    struct aluno a = {"Muriel", 1399999, 0.896};
    printf("A: %s - %d - %f\n", a.nome, a.RA, a.coef);

    //!É possivel atribuir diretamente uma struct em outra.
    //!Somente se as estruturas forem do mesmo tipo.
    //!Cada valor é copiado para seu respectivo campo.
    struct aluno b;
    b = a;
    printf("B: %s - %d - %f\n", b.nome, b.RA, b.coef);

    return 0;
}
```

Definição de Tipo

- Para simplificar os códigos, é possível utilizar o comando *typedef* para definir a estrutura como um tipo de dano.

```
//Definindo um novo nome para a struct aluno
typedef struct aluno Estudante;
```

Definição de Tipo


```c
#include <stdio.h>

//Definindo um novo nome para a struct aluno
typedef struct aluno Estudante;

struct aluno{
    char nome[81];
    int RA;
    float coef;
};

int main(void)
{
    //Estudante é o identificador do tipo de dado de struct aluno
    Estudante a = {"Muriel", 1399999, 0.896};
    printf("A: %s - %d - %f\n", a.nome, a.RA, a.coef);

    return 0;
}
```

Vetores de Struct

- É possível agrupar um conjunto de estruturas (*struct*).
- Cada posição terá seu conjunto de variáveis heterogêneas.

Vetores de Struct


```c
#include <stdio.h>
//Definindo um novo nome para a struct aluno
typedef struct aluno Estudante;

struct aluno{
    char nome[81];
    int RA;
    float coef;
};
```

```c
int main(void)
{
    //Declaração de vetor de struct
    Estudante Aluno[10];
    int i;

    for(i = 0; i < 10; i++)
    {
        scanf(" %[^\n]s", Aluno[i].nome);
        scanf("%d", &Aluno[i].RA);
        scanf("%f", &Aluno[i].coef);
    }

    for(i = 0; i < 10; i++)
    {
        printf("Aluno[%d]:\n", i);
```

Vetores de Struct


```c
#include <stdio.h>
//Definindo um novo nome para a struct aluno
typedef struct aluno Estudante;

struct aluno{
    char nome[81];
    int RA;
    float coef;
};
```

```c
int main(void)
{
    //Declaração de vetor de struct
    Estudante Aluno[10];
    int i;

    for(i = 0; i < 10; i++)
    {
        scanf(" %[^\n]s", Aluno[i].nome);
        scanf("%d", &Aluno[i].RA);
        scanf("%f", &Aluno[i].coef);
    }

    for(i = 0; i < 10; i++)
    {
        printf("Aluno[%d]:\n", i);
```

Cada posição possui seu próprio conjunto de dados (campos).

Vetores de Struct


```c
#include <stdio.h>
//Definindo um novo nome para a struct aluno
typedef struct aluno Estudante;

struct aluno{
    char nome[81];
    int RA;
    float coef;
};
```

```c
int main(void)
{
    //Declaração de vetor de struct
    Estudante Aluno[10];
    int i;

    for(i = 0; i < 10; i++)
    {
        scanf(" %[^\n]s", Aluno[i].nome);
        scanf("%d", &Aluno[i].RA);
        scanf("%f", &Aluno[i].coef);
    }

    for(i = 0; i < 10; i++)
    {
        printf("Aluno[%d]:\n", i);
```

O campo vem após os colchetes *[i]*.

Aninhamento de Structs

- Uma estrutura pode conter outra estrutura como um dos seus campos.

Aninhamento de Structs

- Uma estrutura pode conter outra estrutura como um dos seus campos.
- Exemplo:
  - Um registro de uma pessoa:
    - Nome;
    - RG;
    - CPF;
    - Data de nascimento;
    - Endereço;

Aninhamento de Structs

- Uma estrutura pode conter outra estrutura como um dos seus campos.
- Exemplo:
  - Um registro de uma pessoa:
    - Nome;
    - RG;
    - CPF;
    - Data de nascimento;
    - Endereço;

Data pode ser uma *struct* que contém dia, mês e ano.

Aninhamento de Structs

- Uma estrutura pode conter outra estrutura como um dos seus campos.
- Exemplo:
  - Um registro de uma pessoa:
    - Nome;
    - RG;
    - CPF;
    - Data de nascimento;
    - Endereço;

Endereço pode ser uma *struct* que contém rua, numero, bairro, cep e cidade.

Aninhamento de Structs


```
struct ENDERECO{
     char Rua[50];
     int Numero;
     char Bairro[50];
     char CEP[50];
     char Cidade[50];
     char Estado[50];
};
```

```
struct DATA{
     int dia, mes, ano;
};

struct PESSOA{
     char Nome[50];
     char RG[15];
     char CPF[10];
     struct DATA nasc;
     struct ENDERECO ender;
};
```

Aninhamento de Structs

Variável ***ender*** do tipo ***struct ENDERECO***

```
struct ENDERECO{
     char Rua[50];
     int Numero;
     char Bairro[50];
     char CEP[50];
     char Cidade[50];
     char Estado[50];
};
```

```
struct DATA{
     int dia, mes, ano;
};

struct PESSOA{
     char Nome[50];
     char RG[15];
     char CPF[10];
     struct DATA nasc;
     struct ENDERECO ender;
};
```

Aninhamento de Structs

Variável ***nasc*** do tipo ***struct DATA***

```
struct ENDERECO{
     char Rua[50];
     int Numero;
     char Bairro[50];
     char CEP[50];
     char Cidade[50];
     char Estado[50];
};
```

```
struct DATA{
     int dia, mes, ano;
};

struct PESSOA{
     char Nome[50];
     char RG[15];
     char CPF[10];
     struct DATA nasc;
     struct ENDERECO ender;
};
```

Aninhamento de Structs

```c
int main(void)
{
    struct PESSOA p;

    printf("Informe o nome: ");
    scanf(" %[^\n]s", p.Nome);
    printf("Informe o dia de nascimento: ");
    scanf("%d", &p.nasc.dia);
    printf("Informe o mes de nascimento: ");
    scanf("%d", &p.nasc.mes);
    printf("Informe o ano de nascimento: ");
    scanf("%d", &p.nasc.ano);
    printf("Informe a cidade: ");
    scanf(" %[^\n]s", p.ender.Cidade);

    printf("Nome: %s\n", p.Nome);
    printf("Data: %d/%d/%d\n", p.nasc.dia, p.nasc.mes, p.nasc.ano);
    printf("Cidade: %s\n", p.ender.Cidade);

    return 0;
}
```

Aninhamento de Structs

```c
int main(void)
{
    struct PESSOA p;

    printf("Informe o nome: ");
    scanf(" %[^\n]s", p.Nome);
    printf("Informe o dia de nascimento: ");
    scanf("%d", &p.nasc.dia);
    printf("Informe o mes de nascimento: ");
    scanf("%d", &p.nasc.mes);
    printf("Informe o ano de nascimento: ");
    scanf("%d", &p.nasc.ano);
    printf("Informe a cidade: ");
    scanf(" %[^\n]s", p.ender.Cidade);

    printf("Nome: %s\n", p.Nome);
    printf("Data: %d/%d/%d\n", p.nasc.dia, p.nasc.mes, p.nasc.ano);
    printf("Cidade: %s\n", p.ender.Cidade);

    return 0;
}
```

Acesso segue ordem do aninhamento

Ponteiro de Struct

- Assim como qualquer tipo de dado, o uso de ponteiros segue a mesma regra para as *structs*.

Ponteiro de Struct

- Assim como qualquer tipo de dado, o uso de ponteiros segue a mesma regra para as *structs*.
  - Declarar ponteiro de struct:

```
//Declaração de uma variável
struct aluno variavel;
//Declaração de um ponteiro
struct aluno *ponteiro;
//Atribuindo endereço de uma variável de struct aluno
ponteiro = &variavel;
```

Ponteiro de Struct

- Assim como qualquer tipo de dado, o uso de ponteiros segue a mesma regra para as *structs*.
  - Declarar ponteiro de struct:

```
//Declaração de uma variável
struct aluno variavel;
//Declaração de um ponteiro
struct aluno *ponteiro;
//Atribuindo endereço de uma variável de struct aluno
ponteiro = &variavel;
```

- Um **ponteiro de struct** recebe o endereço de uma **struct do mesmo tipo**.

Ponteiro de Struct

- Os operadores continuam os mesmos.
  - & - endereço de.
  - * - valor de.

Ponteiro de Struct

- Os operadores continuam os mesmos.
  - & - endereço de.
  - * - valor de.

```
struct aluno variavel;
struct aluno *ponteiro;
ponteiro = &variavel;
//Acesso ao valor de cada campo do endereço
//(*ponteiro) == variavel
(*ponteiro).RA = 9999999;
(*ponteiro).coef = 0.90;
strcpy((*ponteiro).nome , "Mazzetto");
```

Ponteiro de Struct

- Os operadores continuam os mesmos.

Cuidado! Graças às regras de precedência, a expressão ***p.campo** equivale a ***(p.campo)** e tem significado muito diferente de **(*p).campo** .

```
(*ponteiro).RA = 9999999;
(*ponteiro).coef = 0.90;
strcpy((*ponteiro).nome , "Mazzetto");
```

Ponteiro de Struct

- Os ponteiros de struct possuem um **operador para abreviar o comando de acesso ao valor.**
  - ‘p->’: acesso ao valor do campo no endereço.
  - Equivale ao uso do conjunto ‘(*p).’

Ponteiro de Struct

- Os ponteiros de struct possuem um **operador para abreviar o comando de acesso ao valor.**
  - ‘p->’: acesso ao valor do campo no endereço.
  - Equivale ao uso do conjunto ‘(*p).’

```
struct aluno variavel;
struct aluno *ponteiro;
ponteiro = &variavel;
//Acesso ao valor de cada campo com ->
//ponteiro-> == (*ponteiro). == variavel.
ponteiro->RA = 8888888;
ponteiro->coef = 0.88;
strcpy(ponteiro->nome , "Muriel Mazzetto");
```

Ponteiro de Struct

- Os ponteiros de struct poss
  abreviar o comando de a
  - 'p->': acesso ao valor do c
  - Equivale ao uso do conjunto

Esse operador existe **apenas para ponteiro de structs.**

```
struct aluno variavel;
struct aluno *ponteiro;
ponteiro = &variavel;
//Acesso ao valor de cada campo com ->
//ponteiro-> == (*ponteiro). == variavel.
ponteiro->RA = 8888888;
ponteiro->coef = 0.88;
strcpy(ponteiro->nome , "Muriel Mazzetto");
```

Struct em Funções

- O uso de struct em funções é como qualquer variável.
  - Passagem por valor ou por referência.
  - Parâmetro de retorno.

Struct em Funções

- O uso de struct em funções é como qualquer variável.
  - Passagem por valor ou por referência.
  - Parâmetro de retorno.

```
typedef struct dma data;

struct dma
{
    int dia, mes, ano;
};
```

Struct em Funções

- O uso de struct em funções é como qualquer variável.
  - Passagem por valor ou por referência.
  - Parâmetro de retorno.

```
typedef struct dma data;

struct dma
{
    int dia, mes, ano;
};
```

```
data diferenca(data d1, data d2)
{
    data aux;
    aux.dia = d1.dia - d2.dia;
    aux.mes = d1.mes - d2.mes;
    aux.ano = d1.ano - d2.ano;
    return aux;
}
```

Alocação Dinâmica de Struct

- A partir do uso de ponteiros é possível **alocar dinamicamente vetores e matrizes de struct.**
- Os comandos seguem o mesmo padrão das variáveis comuns.

Alocação Dinâmica de Struct

- A partir do uso de ponteiros é possível **alocar dinamicamente vetores e matrizes de struct.**
- Os comandos seguem o mesmo padrão das variáveis comuns.
  - Uso de sizeof(struct nome) para descobrir **tamanho da estrutura criada;**
  - Uso de funções da stdlib:
    - Malloc();
    - Calloc();
    - Realloc();
    - Free();

Alocação Dinâmica de Struct

- A partir do uso de ponteiros é possível **alocar**

```
struct PESSOA{
    char Nome[50];
    char RG[15];
    char CPF[10];
    struct DATA nasc;
    struct ENDERECO ender;
};

struct PESSOA* alocar_vetor(int quantidade)
{
    struct PESSOA* vet;
    //Alocar de acordo com o Tipo de Dado da struct
    //Structs aninhadas sao alocadas automaticamente
    vet = (struct PESSOA*) malloc(quantidade * sizeof(struct PESSOA));

    return vet;
}
```

  - Free();

Exercício

- Escreva um código em C que armazene uma lista de pessoas, cada uma contendo:
  - Nome;
  - CPF;
  - Data de nascimento;
- Escreva funções para:
  - Alocar dinamicamente um vetor dessa estrutura;
  - Preencher a lista de contatos;
  - Imprimir a lista de contatos;
  - Buscar um contato específico;