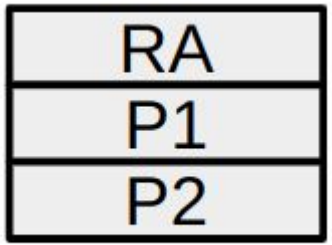
ED1 02: Structs e Ponteiros.

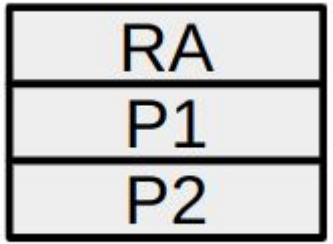
Fabio Irigon Pereira

Structs em C

- Structs permitem agrupar diferentes tipos de dados em uma única unidade.
- São úteis para representar objetos mais complexos, como nós de listas, árvores e grafos.



Structs em C




```c
#include <stdio.h>

struct aluno {
    int ra;
    float p1;
    float p2;
};

int main() {
    struct aluno a1 = {123456, 6.0, 7.5};
    printf("RA: %d, p1: %.2f, p2: %.2f\n" , a1.ra, a1.p1, a1.p2);
    return 0;
}
```

Uso com typedef

```c
typedef struct {
    char nome[50];
    int idade;
} Pessoa;

int main() {
    Pessoa p1 = {"Ana", 30};
    printf("Nome: %s, Idade: %d\n", p1.nome, p1.idade);
    return 0;
}
```

Uso com typedef

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

typedef struct pessoa Pessoa;

int main() {
    Pessoa p1 = {"Ana", 30};
    printf("Nome: %s, Idade: %d\n", p1.nome, p1.idade);
    return 0;
}
```

Exercício

Crie um array de structs pré-preenchida.

Implemente uma função para buscar um elemento da struct.

Ponteiros em C

O que são ponteiros?

- Ponteiros armazenam endereços de memória.
- Permitem manipular dados diretamente na memória, sendo fundamentais para alocação dinâmica e manipulação de **listas, árvores e grafos**.

Ponteiros em C

```
int main() {
    int a = 10;
    int *ptr = &a;   // Ponteiro apontando para 'a'

    printf("Valor de a: %d\n", a);
    printf("Endereço de a: %p\n", &a);
    printf("Valor apontado por ptr: %d\n", *ptr);

    return 0;
}
```

Atenção para os operadores **‘&’** (endereço da variável) e **‘*’** (valor apontado pelo ponteiro.

Structs e Ponteiros juntos

Ponteiros e structs

```c
#include <stdio.h>

void mostrarPessoa(Pessoa *p) {
    printf("Nome: %s, Idade: %d\n", p->nome, p->idade);
}

int main() {
    Pessoa p1 = {"João", 22};
    mostrarPessoa(&p1);   // Passando a struct por referência
    return 0;
}
```

Ponteiros e structs

```c
#include <stdio.h>

void mostrarPessoa(Pessoa *p) {
    printf("Nome: %s, Idade: %d\n", p->nome, p->idade);
}

int main() {
    Pessoa p1 = {"João", 22};
    mostrarPessoa(&p1);   // Passando a struct por referência
    return 0;
}
```

Atenção para o operador ‘**->**’ (valor do campo da estrutura apontada pelo ponteiro).

Alocação dinâmica (malloc)

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

typedef struct {  // cria a struct
    int id;
    char nome[30];
} Aluno;

int main() {
    Aluno *a = (Aluno *)malloc$izeof(Aluno)); // aloca dinamicamente

    if (a == NULL) { // testa se conseguiu alocar
        printf("Erro ao alocar memória\n");
        return 1;
    }

    a->id = 1; // preenche a struct
    snprintf(a->nome, sizeof(a->nome), "Maria");

    printf("ID: %d, Nome: %s\n", a->id, a->nome); // mostra conteúdo

    free(a);  // Libera memória alocada
    return 0;
}
```

Exercício 1

Crie uma struct chamada `Produto` com os campos:

- `char nome[50]`
- `float preco`
- `int estoque`

No `main()`, declare três produtos, preencha seus dados manualmente e imprima as informações na tela.

Faça uma função `void atualizarEstoque(Produto *p, int quantidade)` para atualizar o estoque usando ponteiros.

Exercício 2:

Crie uma struct chamada `Retangulo` com os campos:

- `float largura`
- `float altura`

Implemente as seguintes funções:

- `float calcularArea(Retangulo *r)`: retorna a área do retângulo.
- `float calcularPerimetro(Retangulo *r)`: retorna o perímetro.

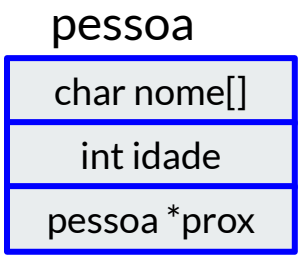
No `main()`, leia os valores do usuário, chame as funções e exiba os resultados.

Structs encadeadas e Ponteiros para ponteiros

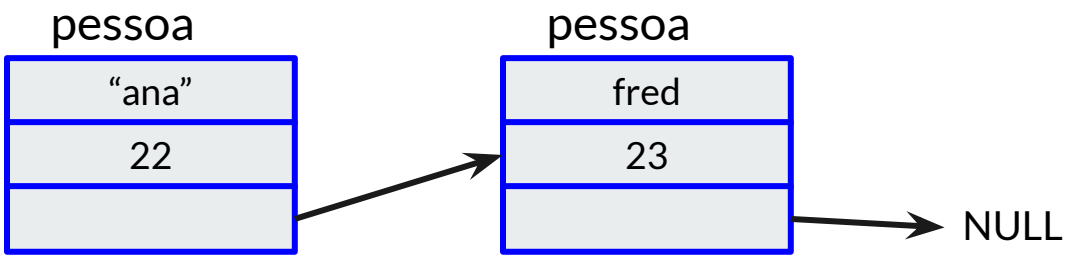
Structs com ponteiros

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

typedef struct pessoa Pessoa;
```



Encadear estruturas



Exercícios próxima aula (importante !)

Crie 5 estruturas do tipo pessoa e ligue uma à outra.

Crie uma função que receba um ponteiro para o primeiro elemento, percorra a lista e imprima cada membro.

Crie uma função que receba um ponteiro para o primeiro elemento, percorra a lista e imprima cada membro.

Crie uma função que receba um ponteiro para o primeiro elemento, e um inteiro ‘n’ e imprima o elemento da posição ‘n’.

Crie uma função que receba um ponteiro para o primeiro elemento e inverta a ordem entre o segundo e o terceiro elementos. (criar ponteiros auxiliares para o 2° e 3° elementos).

Ponteiro para ponteiro para struct

Se um ponteiro é passado por cópia para uma função, seu conteúdo não será modificado.

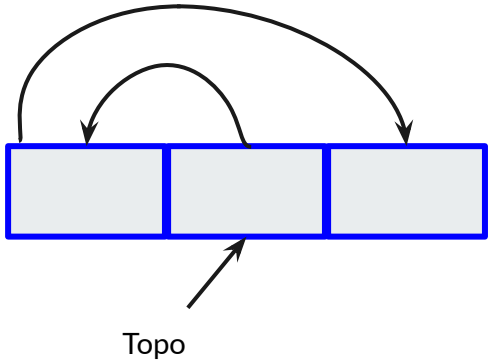
Nestes casos utilizamos dupla indireção.

```
Pessoa aluno;
Pessoa *p = &aluno;
Pessoa **pp = &p;
```

```
printf("%s", aluno.nome);
printf("%s", p->nome);
printf("%s", (*pp)->nome);
```

Exercício:

Crie um array com três nós do tipo Pessoa. Crie um ponteiro chamado topo apontando para o segundo elemento, faça o segundo apontar para o primeiro e o primeiro apontar para o terceiro. Percorra a lista encadeada imprimindo cada elemento.



Exercício (importante):

Crie uma função que receba um ponteiro duplo para o início de uma fila e um ponteiro para um novo elemento e retorne a fila com o novo nó adicionado.

```
void add_member(Pessoa **topo, Pessoa *novo){...}
```