PONTEIROS DE FUNÇÕES

Alunos: *José Vitor, Gustavo Macedo novais*

O que são

Um ponteiro de função em C é uma variável que armazena o endereço de memória de uma função com um determinado tipo de retorno e lista de parâmetros, permitindo que essa função seja chamada indiretamente através do ponteiro.

Declaração

```c
//estrutura da declaração

int (*ponteiro) (int) = nomeDaFunção;
//1 int o que a função devolve
//2 nome do ponteiro
//3 nome da função
//4 presença do parentêses para gatantir que

return 0;
```

Importância

Os ponteiros de funções são importantes porque permitem tornar os programas mais flexíveis e reutilizáveis. Com eles, é possível passar funções como argumentos, escolher qual função executar durante a execução do programa e organizar melhor o código. Essa técnica facilita a criação de programas genéricos e modulares, além de ser bastante utilizada em diversas áreas da programação.

EXEMPLO – CALLBACK

```c
#include <stdio.h>

// Duas funções diferentes:
int soma(int a, int b) {
    return a + b;
}

int subtrai(int a, int b) {
    return a - b;
}

// Função genérica que recebe uma função (callback)
void calcula(int (*operacao)(int, int), int x, int y) {
    int resultado = operacao(x, y);
    printf("Resultado: %d\n", resultado);
}

int main() {
    calcula(soma, 5, 3);      // Passa a função soma como callback
    calcula(subtrai, 5, 3);   // Passa a função subtrai como callback
    return 0;
}
```

Esse código não depende mais do fluxo de dados do desenvolvedor, agora o ambiente de execução decide qual função usar.

EXEMPLO – MENU DE OPÇÕES

```c
#include <stdio.h>

void opcao1() {
    printf("Você escolheu a opção 1\n");
}

void opcao2() {
    printf("Você escolheu a opção 2\n");
}

void opcao3() {
    printf("Você escolheu a opção 3\n");
}

int main() {
    int escolha;

    // Vetor com ponteiros para funções
    void (*menu[3])() = {opcao1, opcao2, opcao3};

    printf("Escolha uma opção (1 a 3): ");
    scanf("%d", &escolha);

    if (escolha >= 1 && escolha <= 3) {
        menu[escolha - 1]();  // Executa a função escolhida
    } else {
        printf("Opção inválida!\n");
    }

    return 0;
}
```

Código mais modular e responsivo, tirando a necessidade de estruturas complexas como if e else

MÉTODOS - STRUCTS

Em linguagens que suportam Programação Orientada a Objetos, como C++ ou Java, é comum associar funções diretamente a objetos, criando os chamados "métodos". No C, que não possui suporte nativo a POO, é possível simular esse comportamento usando ponteiros de funções dentro de structs. Assim, conseguimos criar "métodos" que podem ser chamados a partir de uma struct, tornando o código mais organizado, modular e próximo do estilo orientado a objetos.

```
Cachorro.Latir();
```

EXEMPLO EM C

```c
#include <stdio.h>

// Definindo a struct com "método"
typedef struct {
    char nome[20];
    void (*latir)(const char* nome);
} Cachorro;

// Função que será o "método" da struct
void latido(const char* nome) {
    printf("%s está latindo: Au Au!\n", nome);
}

int main() {
    // Criando o "objeto" cachorro
    Cachorro dog = {"Rex", latido};

    // Chamando o "método" latir
    dog.latir(dog.nome);

    return 0;
}
```

O método latir esta acoplado ao objeto “Cachorro”.

Obrigado!