CURSO SUPERIOR DE ADS

Estruturas

Prof. Fernando Marlon Soares Figueiredo

Disciplina: Programação Estruturada

SECTRAS

ADS

Temas de Miniprojetos – Programação Estruturada (até Structs)

Nível	Tema do Projeto	Descrição	Conteúdos Envolvidos
Fácil	Jogo do Par ou Ímpar	O programa deve permitir que o usuário escolha par ou ímpar, digite um número, e o computador gere outro número aleatório. O sistema soma os valores e exibe quem venceu. Deve oferecer a opção de jogar novamente até que o usuário deseje sair.	Estruturas de decisão (if/else), laços (do-while), uso da função rand(), operadores aritméticos, entrada e saída
Fácil–Médio	Sistema de Notas de Alunos	O programa deve permitir o cadastro do nome de até 10 alunos, suas três notas, e calcular a média final de cada um. Ao final, o sistema deve exibir uma listagem com o nome, média e situação (“Aprovado” ou “Reprovado”).	Vetores, laços (for), estruturas de decisão, printf, scanf, manipulação básica de strings
Médio	Cadastro de Produtos	Desenvolver um sistema simples de estoque com até 20 produtos. Cada produto deve conter nome, quantidade e preço unitário (usando struct). O sistema deve permitir cadastrar, listar e calcular o valor total em estoque.	Struct, vetores, strings, entrada/saída, operadores aritméticos
Médio	Agenda de Contatos	Criar um programa para armazenar até 30 contatos, com nome, telefone e e-mail. O usuário poderá cadastrar, listar e buscar contatos pelo nome. O sistema deve exibir uma mensagem adequada caso o contato não seja encontrado.	Struct, vetores, strings, condicionais e laços

Temas de Miniprojetos – Programação Estruturada (até Structs)

Nível	Tema do Projeto	Descrição	Conteúdos Envolvidos
Médio–Difícil	Sistema de Biblioteca	Desenvolver um sistema que cadastre livros (título, autor e ano). O programa deve permitir: adicionar novos livros, listar todos, buscar por título e exibir o total cadastrado.	Struct, vetores, strings, laços, funções simples
Difícil	Controle de Funcionários	Criar um sistema para registrar funcionários (nome, cargo, salário). O programa deve permitir: adicionar novos funcionários, calcular a média salarial e exibir quem ganha acima da média. Deve incluir menu interativo.	Struct aninhada, vetores, strings, funções, decisão
Difícil	Cadastro de Clientes e Pedidos	O programa deve usar duas structs relacionadas: uma para clientes (nome, CPF) e outra para pedidos (número, valor, cliente associado). Deve permitir cadastrar, listar e exibir o total de pedidos por cliente.	Struct, vetores, strings, entrada/saída, operadores aritméticos
Avançado	Jogo da Forca (versão básica)	O programa deve escolher uma palavra secreta (armazenada em uma string) e permitir que o jogador tente adivinhar letra por letra. Exibir as letras corretas e o número de tentativas restantes. Mostrar mensagem de vitória ou derrota.	Strings, vetores, loops, condicionais, manipulação de caracteres (strlen, strcmp), criatividade

Critério	Descrição	Pontuação Máxima
1. Escolha do tema e nível de desafio	O aluno escolheu um projeto de acordo com seu nível de domínio. Projetos mais “Simples” terão análise mais detalhada e cobrança maior de consistência, mas **todos valem igualmente 5,0 pontos.**	1,0
2. Funcionamento e lógica	O programa deve compilar e executar corretamente, atendendo aos requisitos do projeto escolhido. Erros de lógica ou travamentos reduzem a pontuação.	1,5
3. Estrutura e organização do código	Código devidamente identado, bem comentado e organizado. Uso adequado de funções, vetores, estruturas e boas práticas de programação.	0,5
4. Aplicação dos conteúdos da disciplina	Utilização correta dos elementos vistos até a Aula 10: variáveis, vetores, strings, etc.	0,5
5. Apresentação do projeto	Clareza na explicação do código, demonstrando domínio do que foi implementado.	1,5

Aula de Hoje

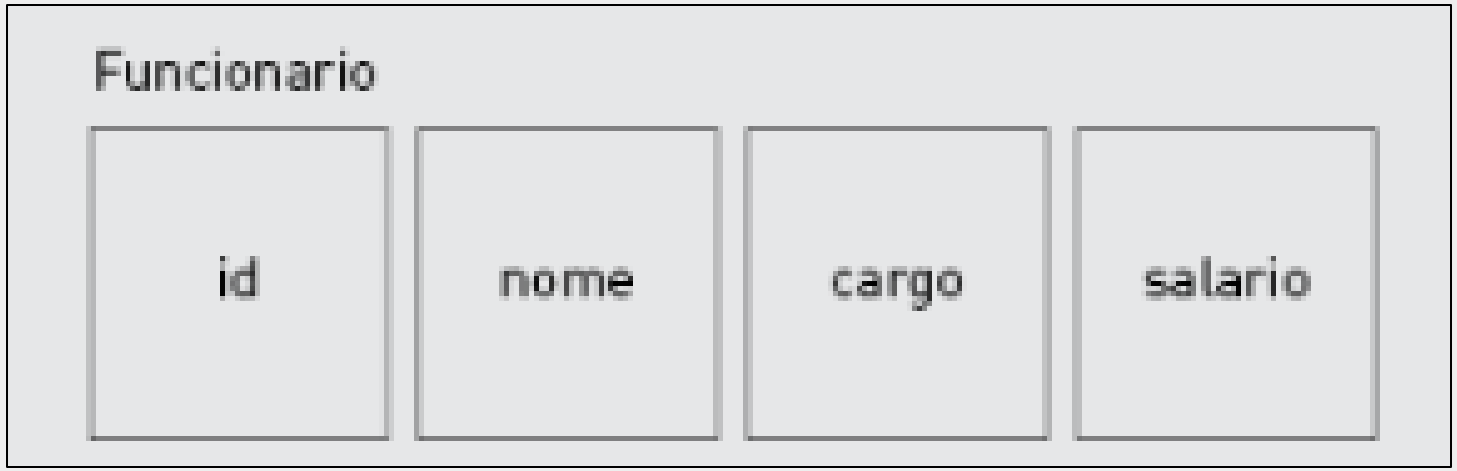
- Conceitos iniciais sobre as estruturas.
- Criando e inicializando valores nas estruturas.
- Arranjos e funções como estruturas.

Introdução

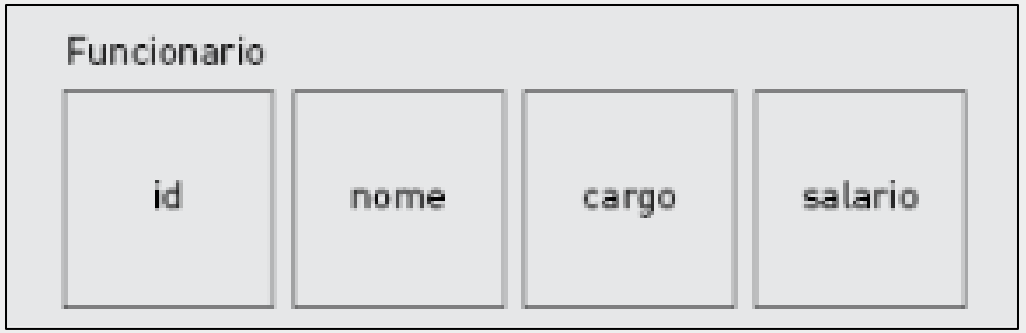
- Os tipos de dados em C podem ser organizados em **básicos** e **estruturados**.
- Tipos de dados básicos são compostos pelos tipos: **int**, **float**, **double** e **char**.
- Tipos estruturados incluem principalmente **arrays**, **ponteiros** e **estruturas**.
- As estruturas apresentam uma forma de juntar diferentes tipos para **construir dados mais complexos**, permitindo organizar informações e, consequentemente, melhorar o código em geral.

Conceitos Básicos

- Estruturas são tipos de dados que permitem agrupar informações de tipos diferentes **em uma única variável**, gerando um novo tipo.
- Estes dados construídos tornam o código mais organizado e legível.
- Um exemplo clássico de estruturas seria criar um tipo de dado para descrever um funcionário de uma empresa...



Vetor de funcionários



- Alguns atributos que funcionários podem ter são **número de identificação**, **nome** e **salário**, por exemplo.
- Caso o programador tivesse que armazenar todos os funcionários da empresa, com o conhecimento visto até então, ele teria que criar um vetor para cada atributo e associá-los pelo índice, tornando o processo confuso e sujeito a falhas.
- Com estruturas, é possível organizar todas essas características em uma única variável, para então criar um vetor do tipo funcionário, onde cada índice possui todas as características de cada funcionário, deixando o programa muito mais organizado.

Sintaxe

Para criar uma estrutura, utiliza-se a seguinte sintaxe:

```
struct Nome

{

tipo1 campo1;

tipo2 campo2;

. . .

tipoN campoN;

};
```

- **Nome**: corresponde ao nome do tipo de dado estrutura, que deve começar com maiúscula por questões de padronização.
- Cada um dos **campos** da estrutura deve ter seu tipo seguido do identificador, que deve seguir a convenção de nomes de variáveis.
- Estruturas devem ser **declaradas no início do código**, entre constantes e protótipos de funções.
- A **criação** de variáveis do tipo da estrutura pode ser feita em **qualquer parte do programa**, evitando que seja uma variável global.

Estrutura geral do programa em C

```
1 /*
2 Comentario inicial descrevendo o programa
3 */
4
5 #include <stdio.h>
6 // outras bibliotecas necessarias
7
8 // constantes com define
9
10 // declaracao de estruturas
11
12 // prototipos de funcoes
13
14 // implementacao de funcoes
```

```
16 int main( ) // programa principal
17 {
18     // declaracoes de variaveis
19
20     // instrucoes
21
22     return 0; // encerramento do programa
23 }
```

Atenção

- É fundamental entender que quando uma estrutura é definida, **estamos criando um tipo de dado, não uma variável.**
- Após a criação da estrutura, caso o programador queira utilizá-la em seu programa, ele deve criar uma variável do tipo da estrutura, assim como é feito com os tipos básicos e com funções.

Criando uma variável de uma struct

- Esta variável deve seguir normalmente as regras de identificadores de variáveis..
- A sintaxe para declaração é:

struct Estrutura variável1, ... variávelN;

Onde Estrutura, corresponde ao nome da estrutura e variável ao nome da variável.

Estamos declarando uma variável, não um ponteiro.

Criando e atribuindo valores

Também podemos inicializar a estrutura os valores da estrutura na sua declaração, de forma semelhante como ocorre com vetores.

struct Estrutura variável1 = {valor **campo1**, ... valor **campoN**};

Acessando os valores dos campos

- O acesso aos campos de uma variável do tipo da estrutura se da com a seguinte sintaxe:

variável1.**campo**;

- A partir do momento em que um campo é acessado, ele se comporta exatamente igual a uma variável.

O que o programa faz?

```c
#include <stdio.h>

struct Funcionario {
    int id;
    char nome[30];
    float salario;
};
```

```c
int main( ) {
    struct Funcionario func;

    printf("Forneca os dados do funcionario: \n");
    printf("ID:");
    scanf("%d", &func.id);
    getchar( );
    printf("Nome:");
    fgets(func.nome, 50, stdin);
    printf("Salario:");
    scanf("%f", &func.salario);
```

```c
    printf("Os dados fornecidos foram: \n");
    printf("ID: %d\n", func.id);
    printf("Nome: %s", func.nome);
    printf("Salario: %f \n", func.salario);

    return 0;
}
```

Pontos Importantes

- As operações realizadas na estrutura devem ser feitas campo a campo, exceto pela operação de atribuição de variáveis diferentes de uma mesma estrutura, que pode ser feita diretamente.
- **Não é possível comparar diretamente duas variáveis de estruturas iguais**, necessitando que as comparações sejam feitas campo a campo.
- Como as estruturas são tipos de dados, todo conteúdo visto sobre arranjos, ponteiros e funções pode ser adaptado para estruturas.

Vetores e Funções com estruturas

```c
#include <stdio.h>

#define MAX_ALUNOS 5

struct Aluno
{
    int matricula;
    char conceito;
};
```

```c
struct Aluno preencheConceito(int numMatr, float nota)
{
    struct Aluno alunoAux;

    alunoAux.matricula = numMatr;

    if(nota >= 6 . 0)
    alunoAux.conceito = 'A';
    else
    alunoAux.conceito = 'R';

    return alunoAux;
}
```

```c
int main()
{
    struct Aluno alunos[MAX_ALUNOS];
    int i, matr;
    float nota;

    for(i =0; i <MAX_ALUNOS; i++)
    {
      printf("Digite a matricula do aluno %d:", i +1);
      scanf("%d", &matr);
      printf("Digite a nota do aluno %d:", i +1);
      scanf("%f ", &nota);
      alunos[i] = preencheConceito(matr, nota);
    }
```

```c
    for(i =0; i <MAX_ALUNOS; i++)
      printf("Aluno %d – Conceito %c. \n", alunos[i]. matricula,
      alunos[i].conceito);

    return 0;
}
```

Ponteiros para estrutura

- Para acessar valores de variáveis do tipo ponteiro para estrutura, costuma-se utilizar os símbolos -> (traço seguido do operador maior que).
- Isto ocorre porque o operador . (ponto) possui maior precedência do que o operador * (asterisco).
- Por exemplo, supondo que a variável **p** seja um ponteiro de uma estrutura que contenha o campo idade, a forma de acessar o valor do campo idade da variável apontada por **p** seria:

```
p->idade //ou (*p).idade
```

Comando typedef

- Este comando fornece um mecanismo para criação de novos nomes (apelidos) para tipos de dados.
- Em estruturas, permite evitar que o programador tenha que escrever o comando **struct** a cada vez que uma estrutura é referenciada, visando melhorar sua legibilidade.
- A sintaxe do comando é:

typedef struct Nome_estrutura_original Nome_estrutura_nova;

Boas Práticas

- A partir do momento que a sintaxe é executada, para o resto do programa, basta referenciar a estrutura pelo seu nome novo em vez da palavra struct seguindo do nome original.
- O uso do typedef é fortemente recomendado para boas práticas de programação.

```
typedef struct

{

tipo 1 campo1;

tipo 2 campo2;

...

tipo N campoN;

} Nome_estrutura;
```

Estruturas com typedef

```
1 #include <stdio.h>
2
3 typedef struct
4 {
5     int dia;
6     int mes;
7     int ano;
8 } Data;
```

```
10 int main( ) {
11     Data d = {7, 9, 1987};
12
13     printf("Data : %d/%d/%d\n", d.dia, d.mes, d.ano);
14
15     return 0;
16 }
```

Exercício 1

- Faça um programa que crie uma estrutura Livro, contendo código e nome como campos.
- Peça ao usuário para preencher as informações de um livro e imprima-as na sequência.

Exercício 1 - Resposta

```c
#include <stdio.h>

typedef struct
{
    int codigo;
    char nome[50];
} Livro;
```

```c
int main( )
{
    Livro liv;

    printf("Digite o codigo do livro:");
    scanf("%d", &liv.codigo);
    getchar();
    printf("Digite o nome do livro:");
    fgets(liv.nome, 50, stdin);

    printf("Informacoes digitadas: \n");
    printf("Codigo: %d\n", liv.codigo);
    printf("Nome: %s \n", liv.nome);

    return 0;
}
```

Exercício 2

- Faça um programa que contenha uma estrutura Cliente, contendo nome (string), CPF (string) e endereço (struct Endereço).
- O campo endereço deve ser dado por outra estrutura chamada Endereço, que deve conter rua e número.
- Crie uma variável do tipo Cliente e preencha seus valores por atribuição.
- Em seguida peça para o usuário digitar o CPF de um cliente e verifique se é equivalente ao número cadastrado.
- Declare a estrutura Endereço antes da estrutura Cliente para que ela possa ser reconhecida.

Exercício 2 - Resposta

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

typedef struct
{
    char rua[50];
    int numero;
} Endereco;

typedef struct
{
    char nome[50];
    char cpf[12];
    Endereco endereco;
} Cliente;

int main( )
{
    Cliente c;
    Endereco e;
    char cpfAux[12];

    strcpy(e.rua, "Rua dos Pampas");
    e.numero = 287;
    strcpy(c.nome, "Mario Silva");
    strcpy(c.cpf, "12345678901");
    c.endereco = e;

    printf("Digite um CPF:");
    fgets(cpfAux, 12, stdin);

    if(strcmp(cpfAux, c.cpf) == 0)
      printf("CPFs iguais. \n");
    else
      printf("CPFs diferentes. \n");

    return 0;
}
```

Exercício 3

- Faça um programa que crie uma estrutura Aluno, contendo nome, número de matrícula e nota.
- No programa principal, crie um vetor de estruturas do tipo Aluno com três elementos e peça para o usuário digitar as informações de cada aluno.
- Em seguida, indique os nomes dos alunos que foram reprovados, ou seja, cujas notas foram menores que 6,0.

Exercício 3 - Resposta

```c
#include <stdio.h>

#define MAX_ALUNOS 3

typedef struct
{
    char nome[50];
    int nMatricula;
    float nota;
} Aluno;

int main( )
{
    Aluno alunos[MAX_ALUNOS];
    int i;

    for(i =0; i <MAX_ALUNOS; i++)
    {
      printf("Digite o nome do aluno %d:", i);
      fgets(alunos[i]. nome, 50, stdin);
      printf("Digite a matricula do aluno %d:", i);
      scanf("%d", &alunos[i].nMatricula);
      printf("Digite a nota do aluno %d:", i);
      scanf("%f", &alunos[i].nota);
      getchar( );
    }
    printf("Alunos reprovados: \n");
    for( i =0; i<MAX_ALUNOS; i++)
    {
      if(alunos[i].nota < 6.0)
        printf("%s \n", alunos[i].nome);
    }

    return 0;
}
```

Exercício 4

- Faça um programa que contenha uma estrutura do tipo Horario que armazena horas, minutos e segundos (inteiros). Desenvolva as seguintes funções e teste-as no programa principal.
- Função que lê um horário (hora, minuto e segundo) fornecido pelo usuário, armazena-o e retorna em uma variável do tipo Horario.
- Função que recebe um parâmetro do tipo Horario e retorna 1 caso o horário seja válido e 0 caso contrário.
- Função que escreve por extenso um tipo de Horario recebido como parâmetro.

Exercício 4 - Resposta

```c
#include <stdio.h>

typedef struct
{
    int hora;
    int minuto;
    int segundo;
} Horario;
```

```c
Horario leHorario( );
int validaHorario(Horario h);
void extensoHorario(Horario h);

Horario leHorario( )
{
    Horario h;

    printf("Digite as horas:");
    scanf("%d", &h.hora);
    printf("Digite os minutos:");
    scanf("%d", &h.minuto);
    printf("Digite os segundos:");
    scanf("%d", &h.segundo);

    return h;
}
```

```c
int validaHorario(Horario h)
{
    int valido =1;

    if(h.hora <0 || h.hora >23 || h.minuto <0 || h.minuto >60 ||
    h.segundo <0 || h.minuto >60)
    valido = 0;

    return valido;
}
```

```c
void extenso Horario(Horario h)
{
    printf("%d horas, %d minutos e %d segundos. \n", h.hora, h.minuto,
    h.segundo);
}
```

Exercício 4 - Resposta continuação

```
43 int main( )
44 {
45     Horario hora;
46
47     hora = leHorario( );
48
49     if(validaHorario(hora) == 1)
50     {
51     printf("Horario valido! \n");
52     extensoHorario(hora);
53     }
54     else
55     printf("Horario invalido! \n");
56
57     return 0;
58 }
```