Professor
Maromo

LINGUAGEM DE PROGRAMAÇÃO

Material 009

C

Github
maromo71

Agenda

Estrutura de um Programa:

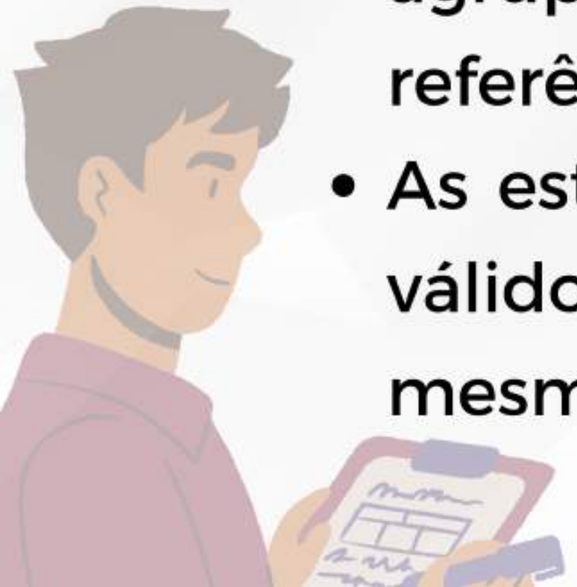
- **Estruturas (Structs) ou Registros**

Material: LP_009

Registros ou Struct

Diz Damas(2007):

- Estruturas em C, permitem colocar, em uma única entidade, elementos de tipos diferentes.
- Uma estrutura é um conjunto de uma ou mais variáveis agrupadas sob um único nome, de forma a facilitar sua referência.
- As estruturas podem conter elementos com qualquer tipo válido em C(tipos básicos, vetores, ponteiros, strings, ou mesmo outras estruturas).



Registros ou Struct
Exemplo

Empregado

Idade

Departamento

Nome

Salario

Estado civil

Registro: Empregado

Para representarmos um empregado, podemos criar uma estrutura e agrupar as variáveis (relacioná-las).

```
struct Empregado {
    int id;
    char nome[100];
    double salario;
};
```

Registro - Sintaxe

```
struct nome_da_estrutura {
    tipo1.. campo1, campo2;
    ...
    tipoN.. campo;
};
```

Declarar uma estrutura corresponde unicamente a definição de um novo tipo, e não à declaração de variáveis do tipo estrutura.

Registro - Exemplo

Uma estrutrura para uma data

```
struct Data{
    int dia, ano;
    char mes[15];
};
```

Declarar uma estrutura corresponde unicamente a definição de um novo tipo, e não à declaração de variáveis do tipo estrutura.

Compilador

A definição de uma estrutura indica que, a partir daquele momento o compilador passa a conhecer um novo tipo, chamado struct Data, que é composto por dois inteiros e um vetor com 15 caracteres.

Ou seja, Data não é uma variável, e sim o nome pelo qual é conhecida essa nova denominação de tipo.

Declaração do tipo criado

Para declarar uma variável do tipo struct Data, basta indicar o tipo seguido do nome da variável.

```
Exs:
struct Data d;
struct Data datas[100];
struct Data *ptr_Data;
```

NOTA:

Em que:

- **d é uma variável do tipo struct Data,**
- **datas é um vetor de 100 elementos sendo cada um deles do tipo struct Data,**
- **ptr_Data é um ponteiro para o tipo struct Data.**

Outra forma de declaração

Pode-se declarar no momento da definição de uma estrutura. Veja:

```
struct Data {
    int dia, ano;
    char mes[21];
} d, datas[100], *ptr_Data;
```

Acesso aos campos membros

Para acessar um campo membro de uma estrutura usa-se o operador ponto [.] desta forma:

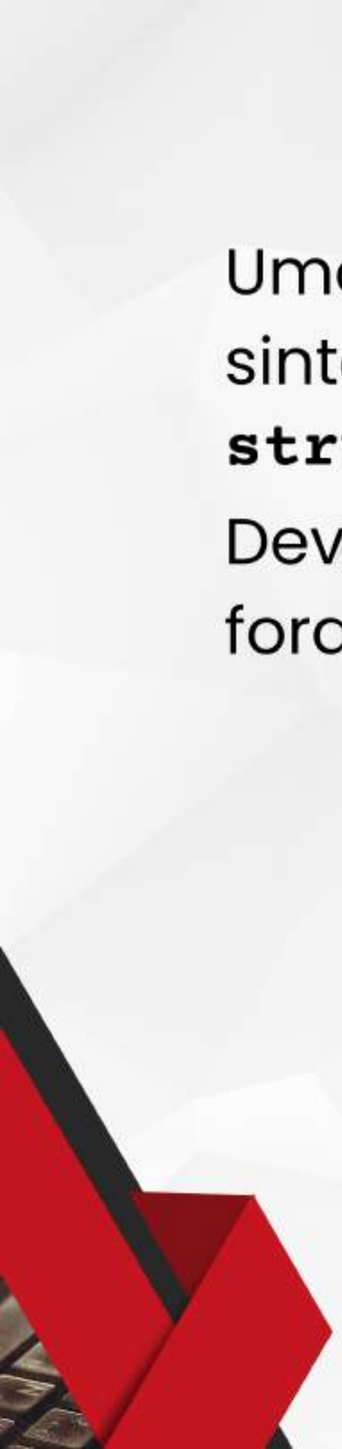
```
var.membro
```

Carga inicial

Uma estrutura pode ser iniciada quando é declarada usando-se a sintaxe:

```
struct nome_struct var = {v1, v2, .., vn}
```

Deve colocar na chave os valores dos membros, pela ordem em que foram definidos na estrutura.



Representação Gráfica [Estrutura Empregado]

```
struct Empregado{
   int matr;
   float salario;
   char nome[20];
};
```

Representa Gráfica

matr	salario	1	2	3.....nome.... 19	20

Exemplo

```c
#include <stdio.h>
#include <stdlib.h>
struct empregado{
 int matr;
 float salario;
 char nome[20];
};
void imprimir(struct empregado e){
  printf("\n\nDADOS CADASTRADOS\n");
  printf("Matricula.....: %d\n", e.matr);
  printf("Salario.......: %5.2f\n", e.salario);
  printf("Nome..........: %s\n", e.nome);
}
int main(){
  struct empregado emp[5];
  printf("\nAlimentando com dados do empregado\n");
  printf("==================================\n");
  printf("Digite a Matricula do Empregado: ");
  scanf("%d", &emp.matr);
  printf("Digite o Salario do Empregado: ");
  scanf("%f", &emp.salario);
  fflush(stdin); // Limpar buffer do teclado.
  printf("Digite o Nome do Empregado: ");
  gets(emp.nome);
  imprimir(emp);
  return 0;
}
```

O código define **uma estrutura chamada empregado com três campos: matrícula, salário e nome.** Uma função **imprimir é criada para mostrar os detalhes de um empregado específico**. No main, **é criado um array emp para armazenar até cinco empregados**, mas apenas um empregado é preenchido com dados do usuário. **Após coletar os detalhes desse empregado (matrícula, salário e nome), a função imprimir é chamada para exibir esses detalhes na tela.**

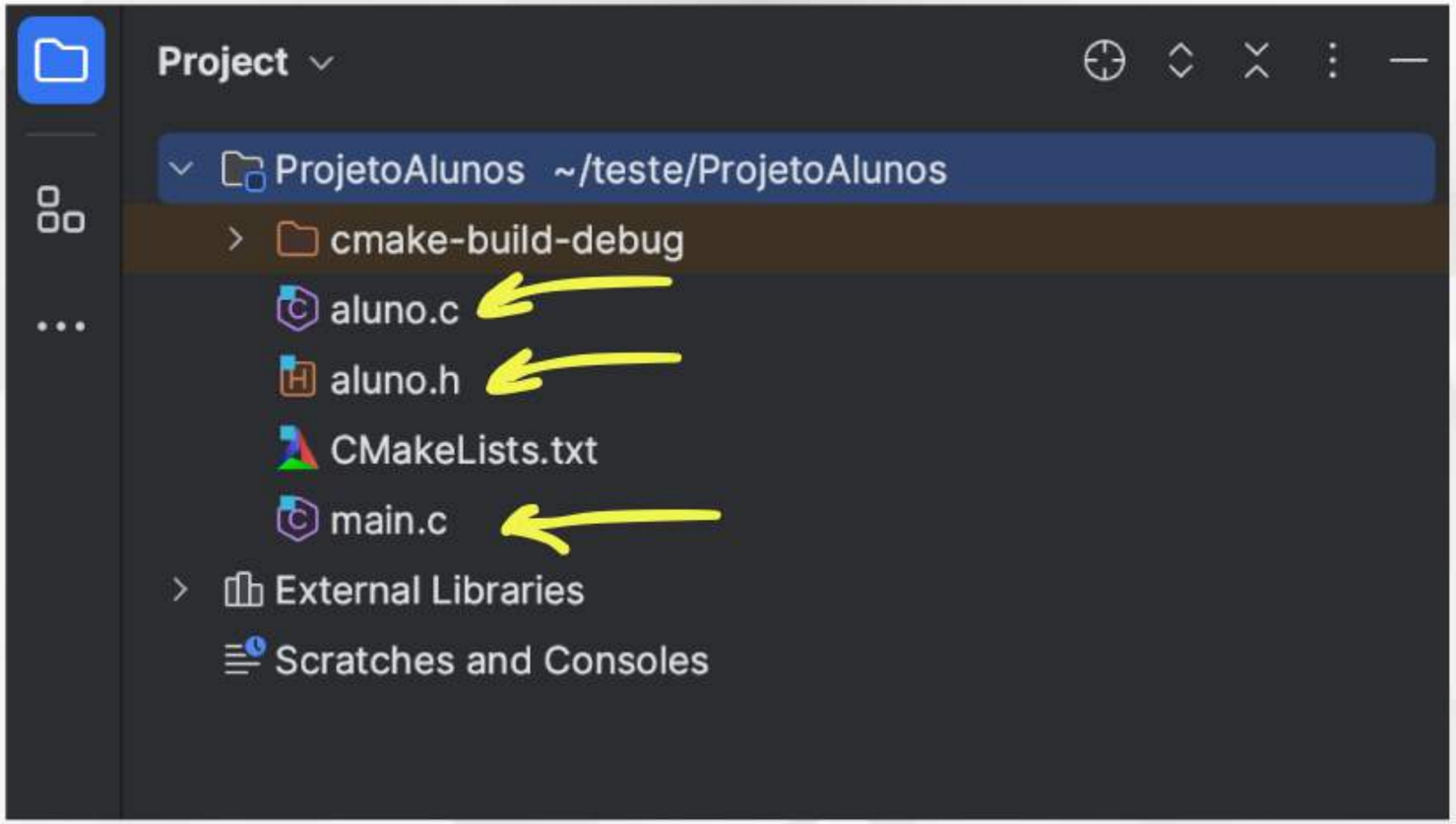
Estruturas Complexas

ProjetoAlunos

Inicie um novo projeto com o nome acima para ilustrarmos o uso de estruturas complexas usando vetor.
Consideremos um exemplo em que desejamos armazenar uma tabela com dados de alunos. Podemos organizar os dados em um vetor.

Três arquivos:

- main.c
- aluno.c
- aluno.h



Estrutura

Nome: cadeia de 80 caracteres.
Matrícula: número inteiro.
Endereço: cadeia com até 120 caracteres.
Telefone: cadeia com até 20 caracteres.
Nota: número de ponto flutuante (double).

Evitando consumo desnecessário de memória

Ao invés de montarmos uma tabela de alunos usando um vetor global com um número máximo de alunos. Usaremos ponteiros, na verdade **um vetor de ponteiros**.

Arquivo aluno.h

O arquivo de cabeçalho a seguir define uma **estrutura nomeada Aluno**, que possui campos para armazenar informações sobre um aluno: nome, matrícula, endereço, telefone e nota. Um **tipo alias aluno** é então definido como um **atalho para struct Aluno**, simplificando a referência a essa estrutura no código.

Em seguida, **um tipo alias p_aluno** é declarado para representar **um ponteiro para a estrutura Aluno**, facilitando a manipulação de ponteiros para essa estrutura. Finalmente, **o vetor v, que pode armazenar até 100 ponteiros para a estrutura Aluno**, é declarado como **extern**, indicando que sua definição real (ou seja, a alocação de memória para o vetor) ocorrerá em outro arquivo de código-fonte.

aluno.h

```
#define MAX 100
typedef struct Aluno {
    char nome[81];
    int matricula;
    char endereco[121];
    char telefone[21];
    double nota;
} aluno;
typedef aluno *p_aluno;

extern p_aluno v[MAX];
```

Completando o aluno.h

Precisamos ainda definir os cabeçalho das funções que serão criadas, uma para inicializar o vetor, de maneiras que os ponteiros fique inicialmente declarados como nulos, uma rotina para ler os dados do teclado e outras duas para imprimir os dados dos alunos cadastrados.

```
....
extern p_aluno v[MAX];

void inicializa();

void ler_dados(int i);

void imprimir_unico(int i);

void imprimir_tudo();
```

Aluno.c

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

void inicializa(){
    int i;
    for(i=0; i<MAX; i++){
        v[i] = NULL;
    }
}
```

```c
void ler_dados(int i){
    if(v[i]==NULL){
        v[i] = (p_aluno)malloc(sizeof(aluno));
        fflush(stdin);
        printf("Digite o nome: \n");
        gets(v[i]->nome);
        printf("Digite a matricula: \n");
        scanf("%d", &v[i]->matricula);
        fflush(stdin);
        printf("Digite o endereco: \n");
        gets(v[i]->endereco);
        fflush(stdin);
        printf("Digite o telefone: \n");
        gets(v[i]->telefone);
        fflush(stdin);
        printf("Digite a nota: \n");
        scanf("%lf", &v[i]->nota);
    }
}
```

Aluno.c

```
void imprimir_unico(int i){
    if(v[i] != NULL){
        printf("Nome:       %s \n", v[i]->nome);
        printf("Matricula: %8d \n", v[i]->matricula);
        printf("Endereco:  %s \n", v[i]->endereco);
        printf("Telefone:  %s \n", v[i]->telefone);
        printf("Nota:       %lf \n", v[i]->nota);
        printf("\n");
    }
}


void imprimir_tudo(){
    int i;
    for(i=0; i<MAX; i++){
        imprimir_unico(i);
    }
}
```

Finalmente o main.c

- O código principal apresenta um sistema simples de gerenciamento de alunos. Nele, é exibido um menu que permite ao usuário escolher entre cadastrar um novo aluno, imprimir os dados de um aluno específico, exibir as informações de todos os alunos ou sair do programa.
- Esse menu é apresentado repetidamente até que o usuário escolha sair.
- Ao optar por cadastrar um novo aluno, a **função ler_dados() é chamada para coletar e armazenar os detalhes do aluno na posição atual**, indicada pela **variável i**.
- Se o usuário quiser visualizar os dados de um aluno específico, ele deve fornecer a posição do aluno, e a função **imprimir_unico() é chamada**.
- Para imprimir os detalhes de todos os alunos cadastrados, a função **imprimir_tudo() é utilizada.** Ao final de cada operação, o programa aguarda um pressionamento de tecla antes de retornar ao menu principal.

main.c

```c
#include <stdio.h>
#include "aluno.h"

p_aluno v[MAX];

int main() {
    int i = 0; //posicao a ser cadastrada
    int opcao = 0;
    do{
        int p=0; // variavel para a posicao de procura
unica
        printf("Menu Principal \n");
        printf("============== \n");
        printf("1. Cadastrar novo aluno \n");
        printf("2. Imprimir dados de unico aluno \n");
        printf("3. Imprimir dados de todos os alunos \n");
        printf("9. Sair do programa \n");
        printf("Digite sua opcao: ");
        scanf("%d", &opcao);

}
switch (opcao) {
        case 1:
            ler_dados(i);
            printf("Tecle algo para continuar \n");
            i++; //passa para a prox posicao
            getchar();
            break;
        case 2:
            printf("Digite a posicao a imprimir ?\n");
            scanf("%d", &p);
            imprimir_unico(p);
            getchar();
            break;
        case 3:
            imprimir_tudo();
            getchar();
            break;
        case 9:
            printf("Fim\n");
            break;
        default:
            printf("Opcao invalida \n");
    }
  }while(opcao !=9);
}
```

Desafio Gerenciamento de Livros

Instruções - Desafio

Exercício: Sistema de Gerenciamento de Livraria

Você foi contratado para desenvolver um sistema simples de gerenciamento para uma pequena livraria. O sistema deve permitir as seguintes operações:

- Cadastrar um novo livro.
- Imprimir os detalhes de um livro específico.
- Imprimir os detalhes de todos os livros no estoque.
- Sair do sistema.

Instruções - Desafio

Estrutura do Livro: Crie uma estrutura chamada Livro que deve conter os seguintes campos:

- Título (string com no máximo 100 caracteres).
- Autor (string com no máximo 50 caracteres).
- ISBN (string com 13 caracteres).
- Preço (número decimal).
- Quantidade em estoque (número inteiro).

Instruções - Desafio

Funções:

- **ler_dados(int posicao):** Esta função deve coletar os detalhes de um novo livro e armazená-lo na posição fornecida no vetor de livros.
- **imprimir_unico(int posicao):** Esta função deve imprimir os detalhes do livro na posição fornecida.
- **imprimir_tudo():** Esta função deve imprimir os detalhes de todos os livros cadastrados.

Instruções - Desafio

Menu Principal:

- O programa deve começar apresentando um menu para o usuário, permitindo que ele escolha uma das operações listadas.
- O menu deve ser repetidamente exibido até que o usuário opte por sair do sistema.
- Use um vetor para armazenar todos os livros cadastrados. Por simplicidade, limite o número máximo de livros a 100.

Instruções - Desafio

Considerações Adicionais:

- Lembre-se de implementar tratamentos adequados para entrada de dados, garantindo que o usuário insira as informações corretamente.
- Considere a possibilidade de o livro já estar cadastrado pelo ISBN e, nesse caso, atualize a quantidade em estoque.

Instruções - Desafio

Dica: Você pode basear-se no código de gerenciamento de alunos fornecido anteriormente como referência, mas lembre-se de adaptá-lo ao contexto de uma livraria e de seguir as instruções.

Desafio: **Como um bônus**, implemente uma função que permita ao usuário pesquisar um livro pelo título ou autor e exibir os detalhes correspondentes.

Referências

DAMAS, L. M. D. Linguagem C. LTC, 2007.

HERBERT, S. C completo e total. 3a. ed. Pearson, 1997.