Marisangila Alves, MSc
marisangila.alves@catolicasc.org.br
marisangila.com.br

Católica de Santa Catarina

2025/1

Estrutura de Dados

Estrutura de Dados
Estruturas Heterogêneas

Sumário

> *Estruturas são grupos de variáveis relacionadas entre si algumas vezes chamadas agregadas sob um nome.*
>
> (Deitel; Deitel, 2011)

> *Em C, uma estrutura é uma coleção de variáveis referenciadas por um nome, fornecendo uma maneira conveniente de se ter informações relacionadas agrupadas.*
>
> (Schildt, 1996)

```c
typedef struct
{
    int id;
    char cpf[12];
    char nome[20];
    char sobrenome[20];
    int idade;
    char data_nascimento[11];
    float altura;
} Pessoa;
```

```c
#include <stdio.h>
struct Pessoa{
    int id;
    char cpf[12];
    char nome[20];
    char sobrenome[20];
    int idade;
    char data_nascimento[11];
    float altura;
};
```

Código 1: Definição de `struct`.

```c
struct Pessoa p;
```

```c
Pessoa p;
```

Código 2: Declaração de `struct`.

```c
int main(int argc, char const *argv[])
{
    Pessoa fulana;
    int id = 0;

    fulana.id = id;
    printf("CPF:\n");
    scanf("%s", fulana.cpf);
    printf("Nome:\n");
    scanf("%s", fulana.nome);
    printf("Sobrenome:\n");
    scanf("%s", fulana.sobrenome);
    printf("Idade:\n");
    scanf("%d", &fulana.idade);
    printf("Data de Nascimento:\n");
    scanf("%s", fulana.data_nascimento);
    printf("Altura:\n");
    scanf("%f", &fulana.altura);
    id++;
    return 0;
}
```

Código 3: Leitura dos campos da estrutura.

```c
int main(int argc, char const *argv[])
{
    Pessoa fulana;

    printf("ID: %d\n", fulana.id);
    printf("CPF: %s\n", fulana.cpf);
    printf("Nome: %s\n", fulana.nome);
    printf("Sobrenome: %s\n", fulana.sobrenome);
    printf("Idade: %d\n", fulana.idade);
    printf("Data de Nascimento: %s\n", fulana.data_nascimento);
    printf("Altura: %.2f\n", fulana.altura);
    return 0;
}
```

Código 4: Exibindo valores armazenados na estrutura.

Manipulção com Vetores

```c
int main(int argc, char const *argv[])
{
    Pessoa fulana[N];
    for (int i = 0; i < N; i++)
    {
        fulana[i].id = i;
        printf("CPF:\n");
        scanf("%s", fulana[i].cpf);
        printf("Nome:\n");
        scanf("%s", fulana[i].nome);
        printf("Sobrenome:\n");
        scanf("%s", fulana[i].sobrenome);
        printf("Idade:\n");
        scanf("%d", &fulana[i].idade);
        printf("Data de Nascimento:\n");
        scanf("%s", fulana[i].data_nascimento);
        printf("Altura:\n");
        scanf("%f", &fulana[i].altura);
    }
    return 0;
}
```

Código 5: Uso de estruturas dentro de vetores.

TAD

- Um **TAD (Tipo Abstrato de Dados)** é uma forma de organizar dados e operações de forma abstrata, separando a interface da implementação.
- Ele define **quais operações** podem ser realizadas sobre os dados, sem expor **como essas operações são implementadas**.
- Permite ao programador pensar em termos de **o que o dado representa e o que pode ser feito com ele**, sem se preocupar com detalhes internos.
- A utilização de TADs facilita a **manutenção, reutilização e segurança** do código.

```c
void inserir(Pessoa pessoas[]){
    for (int i = 0; i < N; i++)
    {
        pessoas[i].id = i;
        printf("CPF:\n");
        scanf("%s", pessoas[i].cpf);
        printf("Nome:\n");
        scanf("%s", pessoas[i].nome);
        printf("Sobrenome:\n");
        scanf("%s", pessoas[i].sobrenome);
        printf("Idade:\n");
        scanf("%d", &pessoas[i].idade);
        printf("Data de Nascimento:\n");
        scanf("%s", pessoas[i].data_nascimento);
        printf("Altura:\n");
        scanf("%f", &pessoas[i].altura);
    }
}
```

Código 6: Inserindo registros.

```c
void listar(Pessoa pessoas[]){
    for (int i = 0; i < N; i++)
    {
        if(pessoas[i].id != VAZIO){
            printf("ID: %d\n", pessoas[i].id);
            printf("CPF: %s\n", pessoas[i].cpf);
            printf("Nome: %s\n", pessoas[i].nome);
            printf("Sobrenome: %s\n", pessoas[i].sobrenome);
            printf("Idade: %d\n", pessoas[i].idade);
            printf("Data de Nascimento: %s\n", pessoas[i].data_nascimento);
            printf("Altura: %.2f\n", pessoas[i].altura);
        }
    }
}
```

Código 7: Listando todos os registros.

```c
int buscar(Pessoa pessoas[], char cpf[]){
    for (int i = 0; i < N; i++)
    {
        if(!strcmp(pessoas[i].cpf, cpf)){
            printf("ID: %d\n", pessoas[i].id);
            printf("CPF: %s\n", pessoas[i].cpf);
            printf("Nome: %s\n", pessoas[i].nome);
            printf("Sobrenome: %s\n", pessoas[i].sobrenome);
            printf("Idade: %d\n", pessoas[i].idade);
            printf("Data de Nascimento: %s\n", pessoas[i].data_nascimento);
            printf("Altura: %.2f\n", pessoas[i].altura);
            return 0;
        }
    }
    printf("Nao encontrado!\n");
    return 1;
}
```

Código 8: Buscando registro único.

```c
int atualizar(Pessoa pessoas[], char cpf[]){
    for (int i = 0; i < N; i++)
    {
        if(!strcmp(pessoas[i].cpf,cpf)){
            printf("CPF:\n");
            scanf("%s", pessoas[i].cpf);
            printf("Nome:\n");
            scanf("%s", pessoas[i].nome);
            printf("Sobrenome:\n");
            scanf("%s", pessoas[i].sobrenome);
            printf("Idade:\n");
            scanf("%d", &pessoas[i].idade);
            printf("Data de Nascimento:\n");
            scanf("%s", pessoas[i].data_nascimento);
            printf("Altura:\n");
            scanf("%f", &pessoas[i].altura);
            return 0;
        }
    }
    printf("Nao encontrado!\n");
    return 1;
}
```

Código 9: Atualizando registro.

```c
int excluir(Pessoa pessoas[], char cpf[]){
    for (int i = 0; i < N; i++)
    {
        if(!strcmp(pessoas[i].cpf, cpf)){
            pessoas[i].id = VAZIO;
            strcpy(pessoas[i].cpf," ");
            strcpy(pessoas[i].nome, " ");
            strcpy(pessoas[i].sobrenome, " ");
            pessoas[i].idade = -1;
            strcpy(pessoas[i].data_nascimento, " ");
            pessoas[i].altura = -1.0;
            printf("Excluido com sucesso!\n");
            return 0;
        }
    }
    printf("Nao encontrado!\n");
    return 1;
}
```

Código 10: Excluindo registro.

Estruturas Aninhadas

```c
typedef struct{
    char rua[100];
    int numero;
    char bairro[20];
    char cep[10];
    char cidade[100];
    char uf[3];
}Endereço;
typedef struct{
    char cpf[12];
    char nome_completo[100];
    Endereço endereco;
}Pessoa;
int main(){
    Pessoa pessoa;
    pessoa.endereco.numero = 1024;
    return 0;
}
```

Código 11: Exemplo de *struct* Pessoa e Endereço.

Exemplos

```c
typedef struct
{
    char nome[50];
    float nota1;
    float nota2;
    float nota3;
}Aluno;
```

Código 12: Exemplo simples de uso de nota com struct.

```c
typedef struct
{
    char nome[50];
    float notas[3];
}Aluno;
```

Código 13: Exemplo com vetor de notas.

```c
typedef struct
{
    char nome[50];
    float notas[3][2]; //notas por disciplina.
}Aluno;
```

Código 14: Exemplo com matriz de notas usando struct.

Tópicos Avançados

União I

> *Uma união é um tipo composto similar a* `struct`*, mas* ***todos os membros compartilham o mesmo espaço de memória****. Ou seja, em uma union, apenas* ***um membro por vez*** *pode armazenar um valor válido.*

```c
#include <stdio.h>
union Numero {
  int i;
  float f;
  char c;
};
int main() {
  union Numero n;
  n.i = 10;
  printf("i = %d | Endereço: %p\n", n.i, &n.i);
  n.f = 3.14;
  printf("f = %.2f | Endereço: %p\n", n.f, &n.f);
  printf("i agora = %f (valor alterado)\n", n.i);
  return 0;
}
```

Código 15: Exemplo básico de union

Atenção!

Modificar um membro da `union` sobrescreve os dados de outros membros. O tamanho da `union` é igual ao tamanho do seu maior membro.

- Economizar memória quando apenas **um membro por vez** é necessário.
- Representar tipos de dados variantes, como em protocolos de rede, arquivos binários ou árvores sintáticas.

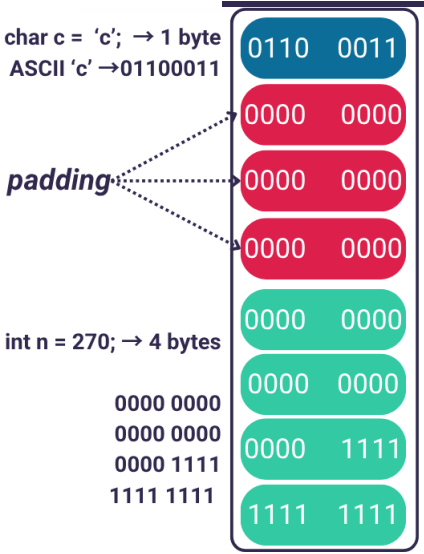
- O alinhamento determina em quais endereços da memória um tipo de dado pode ser armazenado, geralmente múltiplos de seu tamanho ou exigência da arquitetura.
- O padding é o espaço extra que o compilador insere para garantir que os membros de uma **struct** fiquem corretamente alinhados. Mesmo que a soma dos tamanhos dos membros seja pequena, `sizeof(struct)` pode ser maior devido ao padding adicionado pelo compilador.

```c
#include <stdio.h>
#include <stdalign.h>
struct A {
    char c1;      // 1 byte
    int i;        // 4 bytes
    char c2;      // 1 byte
};

struct B {
    char c1; //1 byte
    char c2; //1 byte
    int i; // 4 bytes
};

```

```c
int main() {
    struct A a;
    struct B b;
    printf("Struct A:\n");
    printf("  alignof(A) = %zu\n", alignof(struct A));
    printf("  sizeof(A)  = %zu\n", sizeof(struct A));
    printf("\nStruct B:\n");
    printf("  alignof(B) = %zu\n", alignof(struct B));
    printf("  sizeof(B)  = %zu\n", sizeof(struct B));
    return 0;
}
```

Código 16: Exemplo de alinhamento.

Alinhamento e Padding IV



- **Dicas para otimizar structs:**
  - Ordenar membros do maior para o menor tipo.
  - Agrupar membros do mesmo tipo.
  - Evitar `structs` pequenas dentro de `structs` grandes sem planejar o alinhamento.

Nota:

Pensar no *padding* não muda a lógica do programa, mas pode reduzir o consumo de memória e melhorar a eficiência de acesso.

Um enum é um tipo definido pelo usuário que representa um conjunto de constantes inteiras nomeadas. Ele melhora a legibilidade e a manutenção do código, evitando o uso de números "mágicos".

```c
#include <stdio.h>
typedef enum { DOMINGO, SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO } Dia;
int main() {
    Dia hoje = TERCA;
    if (hoje == DOMINGO || hoje == SABADO)
        printf("Dia de descanso!\n");
    else
        printf("Dia útil!\n");
    return 0;
}
```

Código 17: Exemplo de uso de `enum`

Atenção!

Por padrão, o primeiro valor do enum é 0, e os subsequentes incrementam 1 automaticamente. É possível atribuir valores explícitos aos elementos.

Nota:

Sempre utilize `enums` quando um valor pode assumir **conjunto limitado de estados**, ao invés de usar números ou caracteres diretamente.

(Deitel; Deitel, 2011) - Capítulo 10

DE OLIVEIRA, J.F.; MANZANO, J.A.N.G. **Algoritmos: Lógica para Desenvolvimento de Programação de Computadores**. 16. ed. São Paulo: Editora Érica, 2004.

DE SOUZA, M.A.F. *et al.* **Algoritmos e Lógica de Programação**. São Paulo: Thomson Learning, 2004.

DEITEL, Paul; DEITEL, Harvey. **C: Como Programar**. 6. ed. São Paulo: Pearson Universidades, 2011.

MEDINA, M.; FERTIG, C. **Algoritmos e Programação – Teoria e Prática**. São Paulo: Novatec, 2005.

SCHILDT, Herbert. **C Completo e Total: O Guia Definitivo para Programação em C**. 3. ed. São Paulo: Makron Books, 1996. ISBN 978-8534606928.

Estes slides estão protegidos por uma licença Creative Commons

Este modelo foi adaptado de Maxime Chupin.

Marisangila Alves, MSc
marisangila.alves@catolicasc.org.br
marisangila.com.br

Católica de Santa Catarina

2025/1

Estrutura de Dados

Estrutura de Dados
Estruturas Heterogêneas