Registros (+typedef) e Enumerações

Prof. Fernando Figueira
(adaptado do material do Prof. Rafael Beserra Gomes)

UFRN

Material compilado em 10 de novembro de 2025.

Definição

- Uma implementação pode precisar armazenar diversas variáveis para elementos semelhantes:
- Exemplo:

```
1 //aluno 1
2 char nome1[100];
3 float n1a1, n2a1, n3a1;
4 int matricula1;
5
6 //aluno2
7 char nome2[100];
8 float n1a2, n2a2, n3a2;
9 int matricula2;
```

- Para facilitar a programação, seria interessante agrupar essas variáveis em uma mesma estrutura

Definição

- **Registros** (**struct** em C) é um **tipo de dado composto**
- No exemplo anterior: nome, notas e matrícula vão **compor um novo tipo**

Declaração de struct em C

struct <*identificador*>{
<*campo1*>; //cada variável é um campo da estrutura
<*campo2*>;
...
<*campon*>;
};

```
1  struct Aluno {
2      char nome[100];
3      float n1, n2, n3;
4      int matricula;
5  };
6
7  int main() {
8
9      struct Aluno a1;
10     struct Aluno a2;
11     return 0;
12 }
```

Declaração de struct em C

Observações:

- você pode declarar uma struct dentro de uma função, sendo visível somente nela, mas não é o usual
- você pode definir variáveis dessa estrutura logo após a definição, mas lembre-se de que se tornam globais

```
1 struct Aluno {
2     char nome[100];
3     float n1, n2, n3;
4     int matricula;
5 } aluno1, aluno2; //neste caso ficam globais, evite
```

- você pode inicializar durante a declaração da variável

```
1 struct Aluno a2 = {"Rafael", 5, 6, 7, 201212345};
```

Acessando os campos

Para acessar um campo:
<identificador>.<campo>

```
1  struct Aluno {
2      char nome[100];
3      float n1, n2, n3;
4      int matricula;
5  };
6
7  int main() {
8
9      struct Aluno a1;
10     struct Aluno a2 = {"Rafael", 5, 6, 7, 201212345};
11     a2.n1 = 5.5;
12     a2.n2 = 10.0;
13
14     return 0;
15 }
```

✎ Exercício em sala

- declare uma estrutura **Data** com os campos: dia, mes e ano.
- na função main:
  - declare duas variáveis d1 e d2 do tipo **Data**, inicializando d1 com *06/05/2018*.
  - altere os campos de d2 para *31/12/2018*
  - escreva na tela as duas datas no formato *AAAA/MM/DD* acessando os campos de d1 e d2

Acessando os campos

Uma struct também pode ser um campo de outra struct:

```
1  struct Aluno {
2      char nome[100];
3      float n1, n2, n3;
4      int matricula;
5  };
6
7  struct Turma {
8      struct Aluno alunos[45];
9      int codigo;
10 };
11
12 int main() {
13     struct Turma t1;
14     t1.alunos[0].matricula = 200300000;
15     t1.alunos[0].n1 = 3.0;
16     t1.alunos[0].n2 = 4.5;
17     t1.alunos[0].n3 = 7.0;
18 }
```

✎ Exercício em sala

- utilize o código do exercício anterior
- declare uma estrutura **Tarefa** com os campos dataInicial e dataFinal (ambas do tipo **Data**)
- na função main:
  - declare uma tarefa lista1 do tipo **Tarefa**, com datas *18/02/2018* a *01/04/2018*
  - altere o dia da data inicial para 19
  - altere o mês da data final para 3
  - escreva na tela as duas datas no formato *AAAA/MM/DD*

✎ Exercícios

Implemente as funções que constam em data.c (disponível no repositório) e resolva o exercício na main

✎ Exercício em sala

- declare uma estrutura **Ponto** com os campos: identificador (int), x, y e z (float)
- implemente uma função para calcular a distância euclidiana entre dois pontos
  float distancia(Ponto p1, Ponto p2);
  dada por

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

- implemente uma função para escrever na tela as informações (*ID: (x, y, z)*) sobre um ponto
  void printPonto(Ponto p);

Typedef

Typedef

Typedef permite criar um novo nome (um apelido) para um tipo específico:
typedef *<tipoOriginal><apelido>*;

```
1  struct Aluno {
2      char nome[100];
3      float n1, n2, n3;
4      int matricula;
5  };
6
7  typedef struct Aluno TipoAluno;
8
9  int main() {
10
11     //nao precisa mais escrever struct Aluno!
12     TipoAluno a1 = {"Rafael", 5, 6, 7, 201212345};
13
14     return 0;
15 }
```

Ponteiro para struct

Ponteiro para struct

Similar aos tipos primitivos:

```
1  #include <stdio.h>
2  struct Aluno {
3      char nome[100];
4      float n1, n2, n3;
5      int matricula;
6  };
7
8  int main() {
9
10     struct Aluno *ptr;
11     struct Aluno a2 = {"Rafael", 5, 6, 7, 201212345};
12     ptr = &a2;
13     printf("%s\n", (*ptr).nome);
14
15     return 0;
16 }
```

Ponteiro para struct

Operador ->:
(*id).campo pode ser substituido por id->campo

```c
#include <stdio.h>
struct Aluno {
    char nome[100];
    float n1, n2, n3;
    int matricula;
};

int main() {

    struct Aluno *ptr;
    struct Aluno a2 = {"Rafael", 5, 6, 7, 201212345};
    ptr = &a2;
    printf("%s\n", ptr->nome);

    return 0;
}
```

Enumerações em C

- **Enumerações** (`enum`) são um tipo de dado que permite definir um conjunto de constantes nomeadas
- Útil para melhorar a legibilidade do código e garantir valores válidos
- Sintaxe:

```
1 enum NomeEnum {
2     constante1,
3     constante2,
4     // ...
5 };
```

Exemplo: Dias da Semana

```
1 enum DiasSemana {
2     DOM, SEG, TER, QUA, QUI, SEX, SAB
3 };
4
5 enum DiasSemana hoje = SEG;
6 if (hoje == SEG) {
7     printf("In cio da semana!\n");
8 }
```

- Os valores são atribuídos automaticamente começando em 0
- Podemos atribuir valores específicos: `SEG = 1, TER, QUA, ...`

Definindo um Tipo Booleano

- C não tem um tipo booleano nativo (até C99)
- Podemos criar usando `typedef` + `enum`:

```
1  typedef enum {
2      false,
3      true
4  } bool;
5
6  bool primo(int n) {
7      if (n <= 1) return false;
8      for (int i = 2; i <= n/2; i++) {
9          if (n % i == 0) return false;
10     }
11     return true;
12 }
```

- Agora podemos usar `bool` como tipo e `true`/`false` como valores

✎ Exercício 1: Tipo Booleano

- Modifique a função `primo` do exercício anterior para usar o tipo `bool`
- Crie o tipo `bool` usando `typedef` e `enum`
- Altere o retorno da função para `bool` em vez de `int`
- Teste com alguns valores para verificar se funciona corretamente

✎ Exercício 2: Sistema de Estados

- Crie um `enum` para representar estados de um processo:
  - `INICIAL`
  - `EXECUTANDO`
  - `PAUSADO`
  - `FINALIZADO`
- Crie uma estrutura `Processo` com campos:
  - `id` (int)
  - `estado` (tipo do enum criado)
- Implemente uma função que recebe um `Processo` e imprime seu estado por extenso

Triplos Primos com Lista Ligada

- Na versão otimizada, usamos um vetor para armazenar verificações de primos
- Na próxima aula, implementaremos usando **lista ligada**
- Vantagens:
  - Estrutura dinâmica - não precisa realocar/deslocar elementos
  - Menos escritas na memória
  - Primeiro contato com estruturas de dados além de vetores
- Usaremos **registros** para definir os nós da lista

Estrutura do Nó

```c
typedef struct No {
    int numero;
    bool eh_primo;
    struct No* proximo;
} No;
```

- Cada nó armazena:
  - Um número inteiro
  - Se é primo ou não
  - Ponteiro para o próximo nó