Algoritmos e Programação II

Estruturas Compostas

Prof. Viviane Bonadia dos Santos

Registros

"Um registro é um pacote de variáveis, possivelmente de tipos diferentes. Cada variável é um campo do registro."

"Registros são estruturas capazes de agregar várias informações. Dessa maneira, os programadores podem gerar novos tipos de dados, não se limitando apenas à utilização dos tipos de dados primitivos fornecidos pelas linguagens de programação."

Tipos primitivos: `char`, `int`, `float`...

Registros

"Um registro é um pacote de variáveis, possivelmente de tipos diferentes. Cada variável é um campo do registro."

"Registros são estruturas capazes de agregar várias informações. Dessa maneira, os programadores podem gerar novos tipos de dados, não se limitando apenas à utilização dos tipos de dados primitivos fornecidos pelas linguagens de programação."

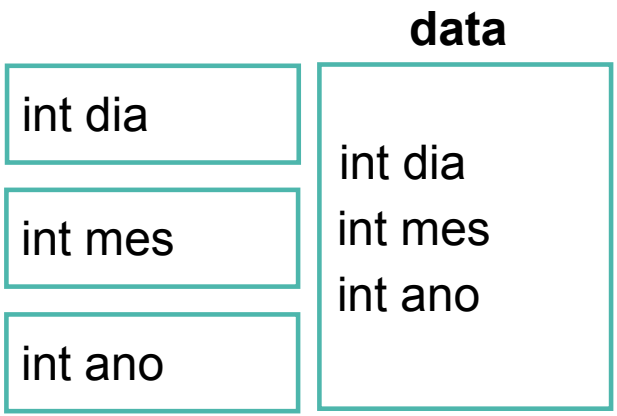
Tipos primitivos: `char`, `int`, `float`...

STRUCT

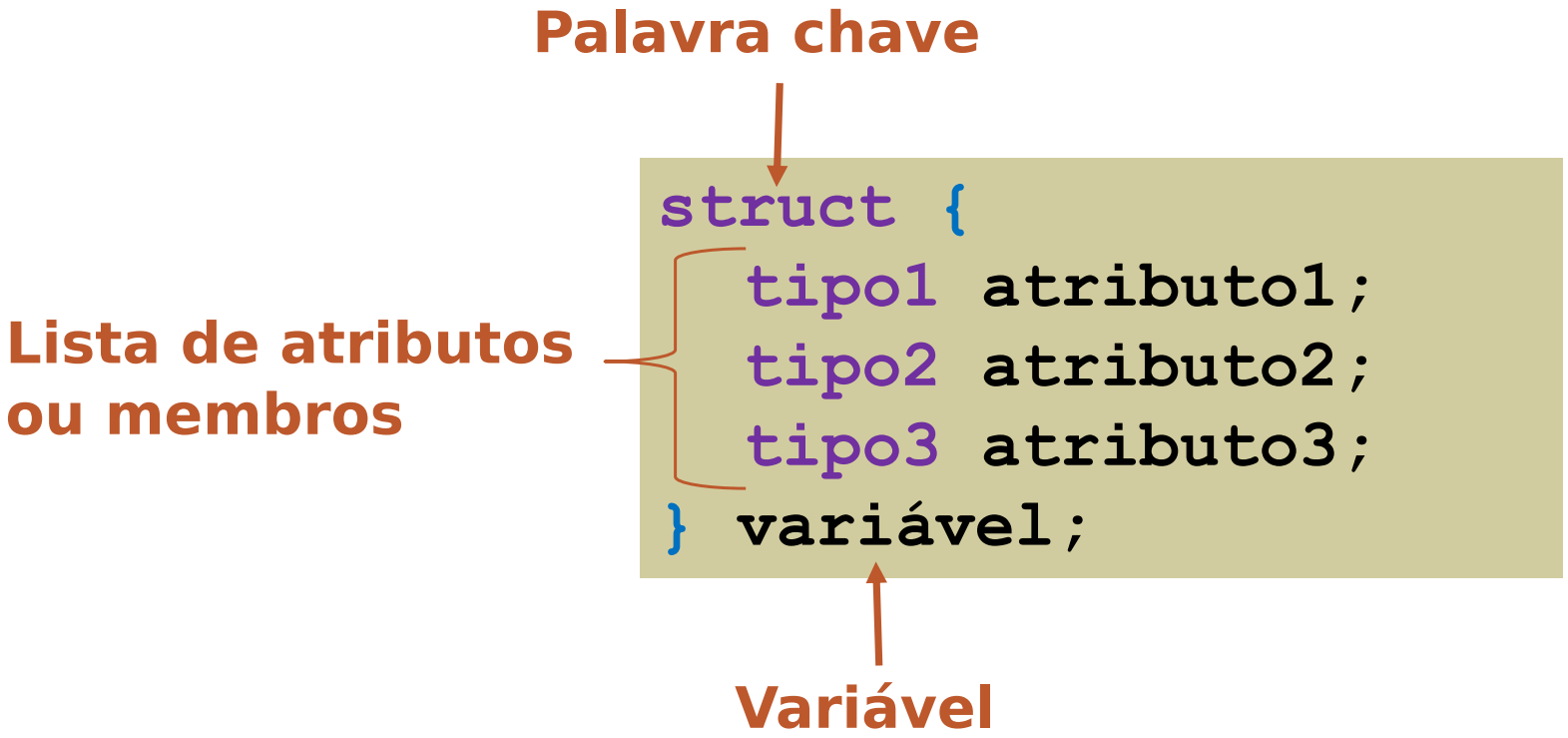
Struct

Na linguagem C, registros são conhecidos como `structs` (o nome é uma abreviatura de *structure*).

- **Motivação:**
  - Agrupar variáveis relacionadas.
  - Organizar dados complexos.
  - Reduzir números de variáveis.
  - Utilizar um único nome de variável.



Declaração



Esta definição cria uma **variável** capaz de armazenar os diferentes atributos definidos (atributo1, atributo2...).
Os membros de uma **variável estrutura** são armazenados em sequência contínua de memória.

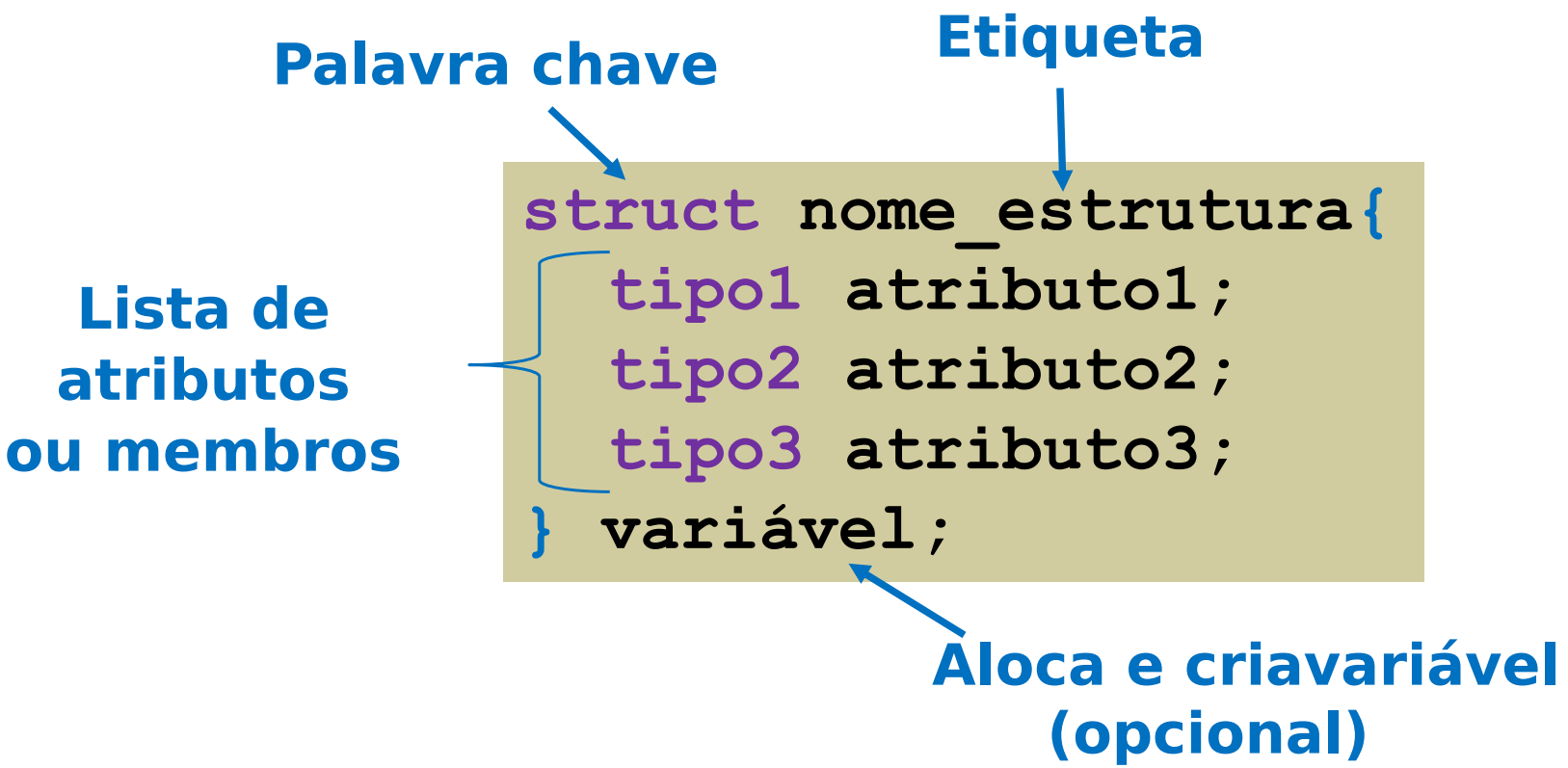
Declaração

O seguinte exemplo declara um registro x com três campos que pode ser usado para armazenar datas:

```
1 struct {
2
3     int dia;
4     int mes;
5     int ano;
6 } x;
```

Declaração

É uma boa ideia dar um nome à classe de todos os registros de um mesmo tipo.



Define um novo tipo de dado!

Declaração

No nosso exemplo, poderíamos dar o nome **dma** para o registro:

```
1  struct dma{
2
3      int dia;
4      int mes;
5      int ano;
6  };
7
8  void main(){
9      struct dma x;   // um registro x do tipo dma
10     struct dma y;   // um registro x do tipo dma
11 }
```

Uma observação importante é que a `struct` só poderá ser utilizada dentro do bloco onde for definida. Para permitir que qualquer parte do seu programa entenda e utilize uma `struct`, sua definição deverá ser feita fora da função `main`, de preferência logo abaixo das `includes`.

Acesso aos atributos

Para se referir a um campo de um registro, basta escrever o nome do registro e o nome do campo separados por um ponto:

- **Acesso:** `x.dia`
- **Atribuição:** `x.dia = 27;`
- **Impressão:** `printf("%d", x.dia);`
- **Leitura:** `scanf("%d", &x.dia);`
- **Copia:** `x = y;`

Acesso aos atributos

```c
#include <stdio.h>

struct dma{
    int dia, mes, ano;
};

int main(){
  struct dma x, y;
  scanf("%d %d %d", &x.dia, &x.mes, &x.ano);
  printf("Dia x = %d. ", x.dia);
  y = x;
  y.dia = x.dia - 1;
  printf("Dia y = %d.", y.dia);
}
```

```
$ ./exemplo
13 04 2016
Dia x = 13. Dia y = 12.
```

Typedef

- Podemos redefinir um tipo de dado dando-lhe um novo nome.
- Para redefinirmos o nome de um tipo de dado usamos o comando **typedef**.

```c
#include <stdio.h>
struct dma{     int dia, mes, ano;};

typedef struct dma data;

int main(){
   data aniversario;
   scanf("%d", &aniversario.dia);
   printf("Dia = %d.\n", aniversario.dia);
}
```

```
$ ./exemplo
30 12 1980
Dia = 30.
```

Typedef

```c
#include <stdio.h>

typedef struct {
    int dia;
    char mes[10];
    int ano;
} data;


int main(){
  data aniversario = {13, "Abril", 2016};

  printf("Mes = %s.\n", aniversario.mes);
}
```

```
$ ./exemplo
Mes = Abril.
```

Declarações aninhadas

- Observe o código abaixo:

```c
#include <stdio.h>

struct aluno{
  char nome[50];
  int dia_matricula;
  int mes_matricula;
  int ano_matricula;
};

int main(){
  struct aluno a1;

  a1.dia_matricula = 10;
}
```

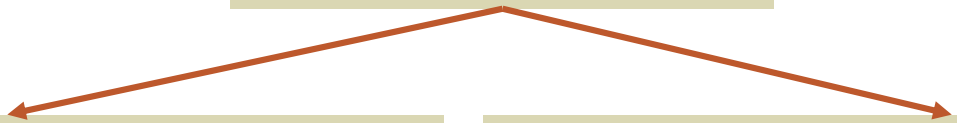
Declarações aninhadas

- As variáveis `dia_matricula`, `mes_matricula` e `ano_matricula` podem ser substituídas por uma estrutura que armazena uma data!

```c
#include <stdio.h>

typedef struct { int dia, mes, ano;} data;

typedef struct {
  char nome[50];
  data matricula;
} aluno;

int main(){
  aluno a1 = {"joao", {2, 2, 2016}};
  a1.matricula.ano = 2020;
  printf("%d %d\n.", a1.matricula.dia, a1.matricula.ano
    );
}
```

Reaproveitando estruturas

```
struct TipoData{
    int dia, mes, ano;
};
```



```
struct TipoAluno{
    char nome[50];
    int codigo_curso;
    struct TipoData data_matricula;
};
```

```
struct TipoFuncionario {
    char nome[50];
    int codigo_departamento;
    struct TipoData data_contratacao;
};
```

UNION

Union

"A sintaxe de definição de novos tipos de dados por meio da palavra **union** é idêntica à da palavra **struct**."

"Entretanto, uma **union** utiliza um mesmo espaço de memória a ser compartilhado com diferentes membros, enquanto uma **struct** aloca um espaço diferente de memória para cada membro."

"Somente um dentre vários valores ocupa a memória. Quando você declara uma variável do tipo **union**, automaticamente é alocado um espaço de memória suficiente para conter o seu maior membro."

"Uniões podem ser usadas para poupar memória."

Union

"A sintaxe de definição de novos tipos de dados por meio da palavra **union** é idêntica à da palavra **struct**."

"Entretanto, uma **union** utiliza um mesmo espaço de memória a ser compartilhado com diferentes membros, enquanto uma **struct** aloca um espaço diferente de memória para cada membro."

"Somente um dentre vários valores ocupa a memória. Quando você declara uma variável do tipo **union**, automaticamente é alocado um espaço de memória suficiente para conter o seu maior membro."

"Uniões podem ser usadas para poupar memória."

Union

"A sintaxe de definição de novos tipos de dados por meio da palavra **union** é idêntica à da palavra **struct**."

"Entretanto, uma **union** utiliza um mesmo espaço de memória a ser compartilhado com diferentes membros, enquanto uma **struct** aloca um espaço diferente de memória para cada membro."

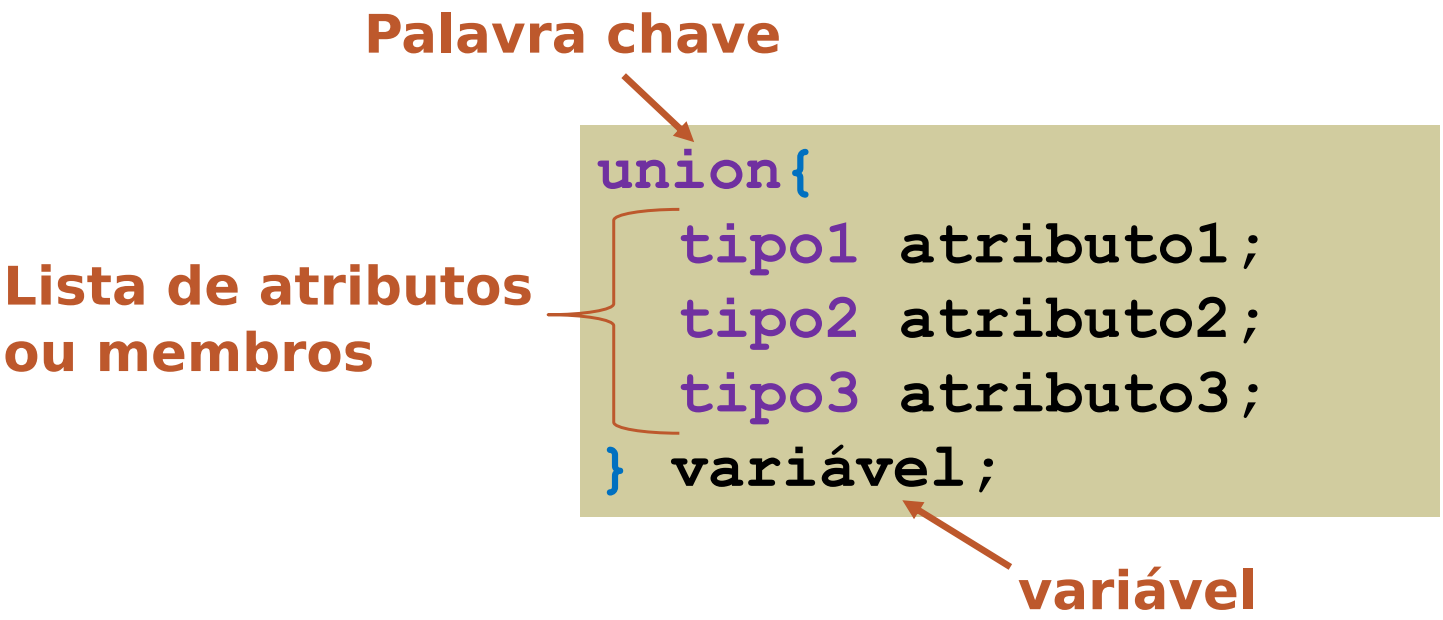
"Somente um dentre vários valores ocupa a memória. Quando você declara uma variável do tipo **union**, automaticamente é alocado um espaço de memória suficiente para conter o seu maior membro."

"Uniões podem ser usadas para poupar memória."

Union

"A sintaxe de definição de novos tipos de dados por meio da palavra **union** é idêntica à da palavra **struct**."
"Entretanto, uma **union** utiliza um mesmo espaço de memória a ser compartilhado com diferentes membros, enquanto uma **struct** aloca um espaço diferente de memória para cada membro."
"Somente um dentre vários valores ocupa a memória. Quando você declara uma variável do tipo **union**, automaticamente é alocado um espaço de memória suficiente para conter o seu maior membro."
"Uniões podem ser usadas para poupar memória."

Union


Palavra chave
union{
tipo1 atributo1;
tipo2 atributo2;
tipo3 atributo3;
} variável;
Lista de atributos
ou membros
variável

Union

```c
#include <stdio.h>

typedef union {
    int inteiro;
    float fracionario;
} Numero;

int main(){
  Numero num;
}
```

Memória

O que tem armazenado na memória??

Union

```c
#include <stdio.h>

typedef union {
    int inteiro;
    float fracionario;

} Numero;

int main(){
  Numero num;
  num.inteiro = 10;
}
```

Memória

O que tem armazenado na memória??

O número inteiro 10.

Union

```c
#include <stdio.h>

typedef union {
    int inteiro;
    float fracionario;

} Numero;

int main(){
  Numero num;
  num.inteiro = 10;
}
```

Memória

O que tem armazenado na memória??

O número inteiro 10.

Union

```c
#include <stdio.h>

typedef union {
    int inteiro;
    float fracionario;

} Numero;

int main(){
  Numero num;
  num.inteiro = 10;
  num.fracionario = 10.5;
}
```

Memória

O que tem armazenado na memória??

O número fracionário 10.5.

Union

```c
#include <stdio.h>

typedef union {
    int inteiro;
    float fracionario;

} Numero;

int main(){
  Numero num;
  num.inteiro = 10;
  num.fracionario = 10.5;
}
```

Memória

O que tem armazenado na memória??

O número fracionário 10.5.

Enumerações

"As enumerações são um tipo de dados através do qual se pode representar um único valor de um pequeno conjunto discreto e finito de alternativas."

- **Conceitos:**
  - Enumeração: conjunto de nomes.
  - Associa internamente cada nome a um número.
  - Valor da variável tipo enumeração é sempre apenas uma das opções.

Enum

Declarção:

```
enum nome_enumeracao {
        opcao1,
        opcao2,
        opcao3,
} variavel;
```

Enum

Declaração:

```
enum {dinheiro[0], cheque[1], vale_refeicao[2],
      cartao[3] } forma_pagamento;
```

Seleção:

```
switch (forma_pagamento) {
        case dinheiro: ...
               break;
        case cheque:  ...
               break;
        case vale_refeicao: ...
               break;
        case cartao: ...
               break;        }
```

A palavra **enum** enumera a lista de nomes automaticamente, dando-lhes números em sequência. A vantagem é que utilizamos esses nomes no lugar de números, o que torna o programa mais claro.

Enum

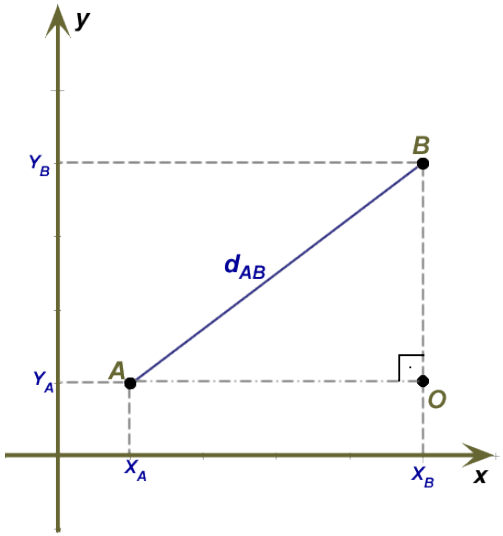
Exemplo:

```c
#include<stdio.h>
typedef enum {jan=1, fev, mar, abr, mai, jun, jul, ago,
     set, out, nov, dez} Mes;

void main(){
  Mes m1, m2, m3;

  m1 = abr; //atribui valores
  m2 = jun;
  m3  = m2 - m1; //operacoes aritmeticas permitidas

  if(m1 < m2) //comparacoes permitidas
    printf("O mes %d vem antes do mes %d.\n", m1, m2);

  if(m2 == jun)
    printf("Junho!\n");

}
```

EXERCÍCIOS

Exercício

- Escreva um programa que recebe dois pontos e calcula a distância entre eles.



$$d_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

Exercício

- Escrever um programa que cadastre médicos de plantão e possua métodos que:
  1. Cadastre novos médicos (suponha que este hospital aceita até 30 médicos de plantão).
  2. Dado um horário liste todos os médicos que estão de plantão neste horário.
  3. Dado um horário e uma especialidade, liste os médicos da especialidade desejada que estão de plantão.

Médicos de plantão

Matrícula	Nome	Hora inicial	Hora final	Especialidade
18001	Rilmar Gomes	18:00	00:00	Clínico Geral
27653	Fernando Soares	00:00	06:00	Dermatologia
19753	Manoel Soares	06:00	18:00	Cardiologia

Algumas referências...

- Treinamento em Linguagem C. Victorine Viviane Mazrahi. Capítulo 8.
- Fundamentos da programação de computadores. Ana Fernanda Gomes Ascencio e Edilene Aparecida Veneruchi de Campos. Capítulo 10.
- http://www.ic.unicamp.br/∼mc102
- http://www.ime.usp.br/∼pf/algoritmos/aulas/stru.html