Vetores, Matrizes e Registros

Graziela Araújo

(aula baseada no material do Prof Fábio Viduani)

Extensão do uso de registros

- Vetor de registros
- Matriz de registros
- Registro contendo como campo
  - Um tipo primitivo
  - Um vetor
  - Outro registro

Variáveis compostas homogêneas de registros

- podemos declarar um **vetor** com identificador **cronometro** como mostrado a seguir:

```
struct {
    int horas;
    int minutos;
    int segundos;
} cronometro[10];
```

  - O vetor **cronometro** contém 10 compartimentos de memória, sendo que cada um deles é do tipo registro. Cada registro contém os campos **horas** , **minutos** e **segundos** , do tipo inteiro.

Variáveis compostas homogêneas de registros

- ou ainda declarar, por exemplo, uma **matriz** com identificador **agenda** como abaixo:

```
struct {
    int horas;
    int minutos;
    int segundos;
} agenda[10][30];
```

  - a matriz **agenda** é uma matriz contendo 10 linhas e 30 colunas, onde cada compartimento contém um registro com os mesmos campos do tipo inteiro **horas** , **minutos** e **segundos** .

Variáveis compostas homogêneas de registros

- poderiam ter sido declaradas em conjunto, numa mesma sentença de declaração:

```
struct {
    int horas;
    int minutos;
    int segundos;
} cronometro[10], agenda[10][30];
```

- ou ainda

```
struct tipoHora{
    int horas;
    int minutos;
    int segundos;
};
...
tipoHora cronometro[10], agenda[10][30];
```

Variáveis compostas homogêneas de registros

- A declaração do vetor **cronometro** tem um efeito na memória que pode ser ilustrado como na figura 16.1.

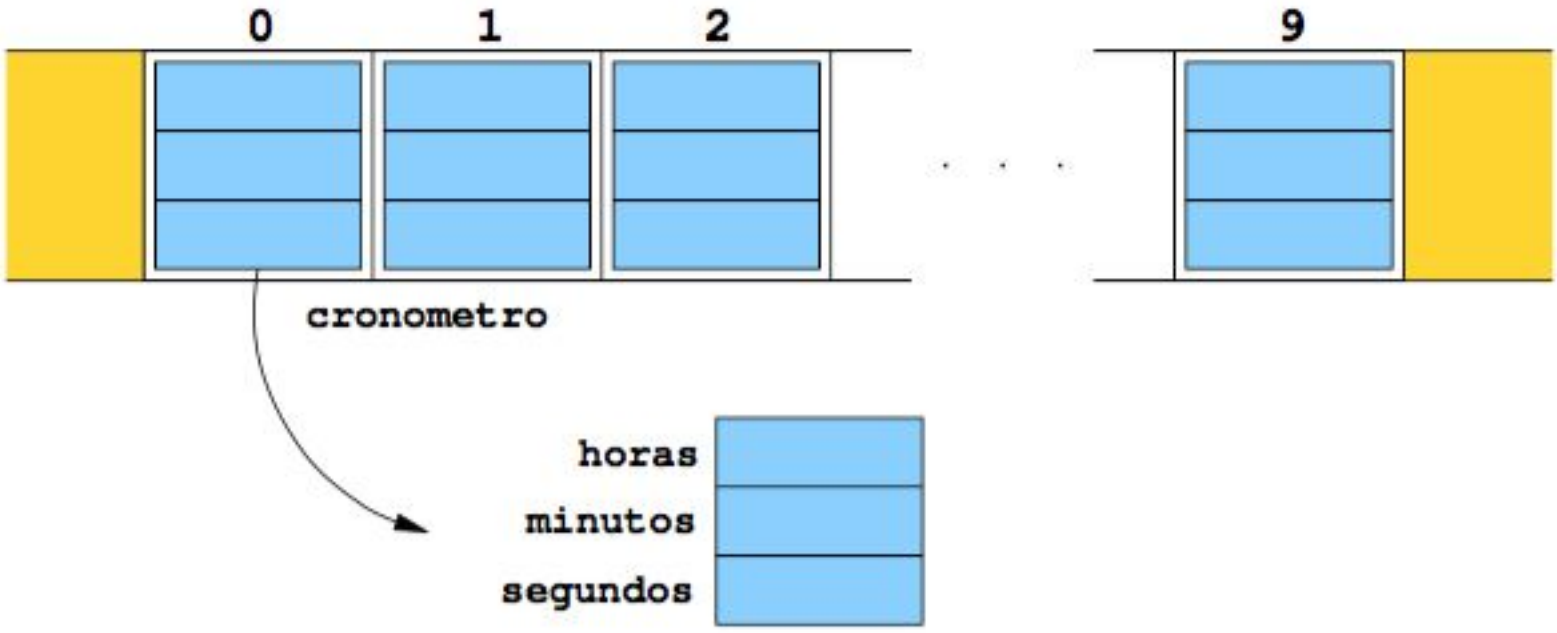

Figura 16.1: Efeito da declaração do vetor `cronometro` na memória.

Variáveis compostas homogêneas de registros

- Atribuições de valores aos campos do registro na posição 0 do vetor **cronometro** :

```
cronometro[0].horas = 20;
cronometro[0].minutos = 39;
cronometro[0].segundos = 18;
```

Variáveis compostas homogêneas de registros

- podemos fazer a atribuição direta de registros para registros declarados da mesma maneira:

```
struct tipoHora{
   int horas;
   int minutos;
   int segundos;
};
...
struct tipoHora cronometro[10], aux;
```

- então as atribuições a seguir são válidas e realizam a troca dos conteúdos das posições **i** e **j** do vetor de registros **cronometro** :

```
aux = cronometro[i];
cronometro[i] = cronometro[j];
cronometro[j] = aux;
```

Variáveis compostas homogêneas de registros

- Observe que todos os campos de um registro são atualizados automaticamente quando da atribuição de um registro a outro registro, podemos fazer:

```
cronometro[i] = cronometro[j];
```

- Ao invés de:

```
cronometro[i].horas = cronometro[j].horas;
cronometro[i].minutos = cronometro[j].minutos;
cronometro[i].segundos = cronometro[j].segundos;
```

PROBLEMA

- Dado um número inteiro *n*, com $1 \leq n \leq 100$, e *n* medidas de tempo dadas em horas, minutos e segundos, distintas duas a duas, ordenar essas medidas de tempo em ordem crescente.
- O programa que será mostrado soluciona o problema acima usando o método de ordenação das trocas sucessivas ou método da bolha.

j j+1

50 40 30 20 10

j j+1

40 50 30 20 10

j j+1

40 30 50 20 10

j j+1

40 30 20 50 10

40 30 20 10 50

---

40 30 20 10 50

30 40 20 10 50

30 20 40 10 50

30 20 10 40 50

---

30 20 10 40 50

20 30 10 40 50

20 10 30 40 50

---

20 10 30 40 50

10 20 30 40 50

j j+1

04:45:59 03:45:59

j j+1

03:45:50 03:45:57

```c
#include <stdio.h>
#define MAX 100
/* Recebe um inteiro n, 1 <= n <= 100, e n medidas de tempo
hh:mm:ss, e mostra esses tempos em ordem crescente */
struct tipoHora
{
    int hh;
    int mm;
    int ss;
};

// funcao principal
int main(void){

    int i, j, n;
    tipoHora cron[MAX], aux;

    printf("Informe a quantidade de medidas de tempo: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("Informe uma medida de tempo (hh:mm:ss): ");
        scanf("%d:%d:%d", &cron[i].hh, &cron[i].mm, &cron[i].ss);
    }
```

```
for (i = n-1; i > 0; i--) {
    for (j = 0; j < i; j++) {
        if (cron[j].hh > cron[j+1].hh) {
            aux = cron[j];
            cron[j] = cron[j+1];
            cron[j+1] = aux;
        }
        else if (cron[j].hh == cron[j+1].hh) {
            if (cron[j].mm > cron[j+1].mm) {
                aux = cron[j];
                cron[j] = cron[j+1];
                cron[j+1] = aux;
            }
            else if (cron[j].mm == cron[j+1].mm) {
                if (cron[j].ss > cron[j+1].ss) {
                    aux = cron[j];
                    cron[j] = cron[j+1];
                    cron[j+1] = aux;
                }
            }
        }
    }
}
```

```
    printf("\nHorários em ordem crescente\n");
    for (i = 0; i < n; i++)
        printf("%d:%d:%d\n", cron[i].hh, cron[i].mm, cron[i].ss);
    return 0;
}
```

Cadeia de caracteres (string)

```
char    str[10] = "formiga";
```

f	o	r	m	i	g	a	\0		
0	1	2	3	4	5	6	7	8	9

```
scanf(" %s", str);    // suponha entrada "gato"
```

g	a	t	o	\0					
0	1	2	3	4	5	6	7	8	9

Registros contendo variáveis compostas homogêneas

- exemplo bastante comum é a declaração de um vetor de caracteres, ou uma cadeia de caracteres, dentro de um registro

```
struct {
   int dias;
   char nome[4];
} mes, aux;
```

- podemos fazer a seguinte atribuição válida ao registro **mes** :

```
mes.dias = 31;
mes.nome[0] = 'J';
mes.nome[1] = 'a';
mes.nome[2] = 'n';
mes.nome[3] = '\0';
```

strcpy(mes.nome, "Jan");

Registros contendo variáveis compostas homogêneas

- Os efeitos da declaração da variável **mes** na memória e da atribuição de valores acima podem ser vistos na figura 16.2

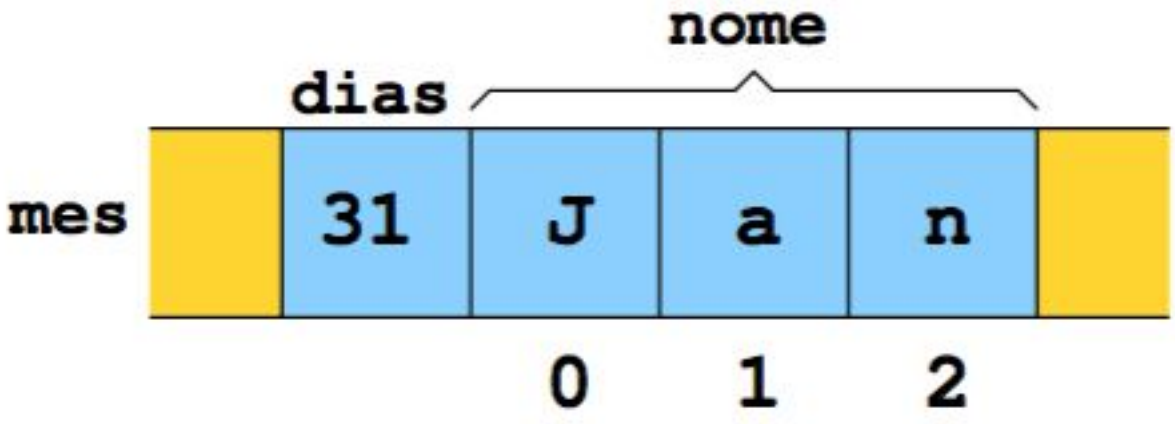

Figura 16.2: Variável `mes` na memória.

Registros contendo variáveis compostas homogêneas

- as regras para cópias de registros permanecem as mesmas, mesmo que um campo de um desses registros seja uma variável composta. Assim, a cópia abaixo é perfeitamente válida:

```
struct {
    int dias,
    char nome[3];
} mes, aux;
```

```
aux = mes;
```

Problema

- Dadas duas descrições de tarefas e seus horários de início no formato hh:mm:ss, escreva um programa que verifica qual das duas tarefas será iniciada antes. Considere que a descrição de uma tarefa tenha no máximo 50 caracteres.

```c
#include <stdio.h>
/* Recebe a descrição e o horário de início de duas atividades, no formato
hh:mm:ss, e verifica qual delas será realizada primeiro */
struct tipoAtividade{
    int horas;
    int minutos;
    int segundos;
    char descricao[51];
};
int main()
{
    int tempo1, tempo2;
    tipoAtividade t1, t2;

    printf("Informe a descrição da primeira atividade: ");
    scanf("%[^\n]", t1.descricao);
    printf("Informe o horário de início dessa atividade (hh:mm:ss): ");
    scanf("%d:%d:%d", &t1.horas, &t1.minutos, &t1.segundos);
    printf("Informe a descrição da segunda atividade: ");
    scanf(" %[^\n]", t2.descricao);
    printf("Informe o horário de início dessa atividade (hh:mm:ss): ");
    scanf("%d:%d:%d", &t2.horas, &t2.minutos, &t2.segundos);
```

```c
int main()
{
    int tempo1, tempo2;
    tipoAtividade t1, t2;

    printf("Informe a descrição da primeira atividade: ");
    scanf("%[^\n]", t1.descricao);
    printf("Informe o horário de início dessa atividade (hh:mm:ss): ");
    scanf("%d:%d:%d", &t1.horas, &t1.minutos, &t1.segundos);
    printf("Informe a descrição da segunda atividade: ");
    scanf(" %[^\n]", t2.descricao);
    printf("Informe o horário de início dessa atividade (hh:mm:ss): ");
    scanf("%d:%d:%d", &t2.horas, &t2.minutos, &t2.segundos);

    tempo1 = t1.horas * 3600 + t1.minutos * 60 + t1.segundos;
    tempo2 = t2.horas * 3600 + t2.minutos * 60 + t2.segundos;

    if (tempo1 <= tempo2)
        printf("%s será realizada antes de %s\n", t1.descricao, t2.descricao);
    else
        printf("%s será realizada antes de %s\n", t2.descricao, t1.descricao);
    return 0;
}
```

Registros contendo registros

```
struct {
   int rga;
   char nome[51];
   struct {
      int dia, mes, ano;
   }nasc;
}estudante;
```

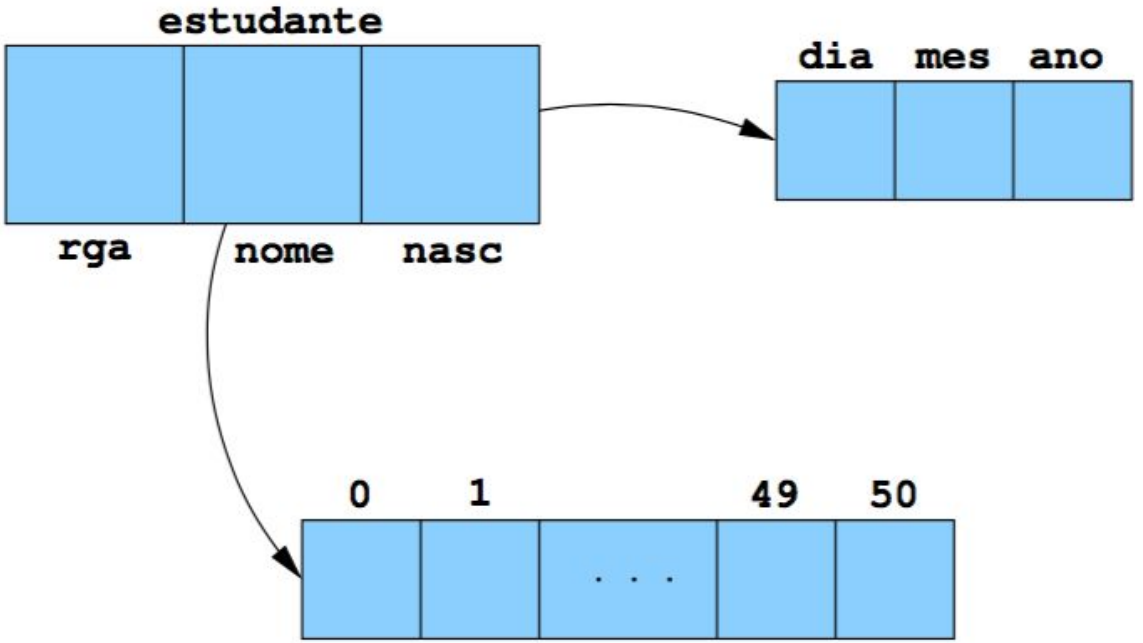

Figura 16.3: Efeitos da declaração do registro `estudante` na memória.

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

struct tipoEstudante{
      int rga;
      char nome[51];
      tipoData nasc;
};
tipoEstudante estudante;
```

Registros contendo registros

- Um exemplo do uso do registro **estudante** e de seus campos é dado a seguir através de atribuições de valores a cada um de seus campos:

```
estudante.rga = 200790111;
estudante.nome[0] = 'J';
estudante.nome[1] = 'o';
estudante.nome[2] = 's';
estudante.nome[3] = 'e';
estudante.nome[4] = '\0';
estudante.nasc.dia = 22;
estudante.nasc.mes = 2;
estudante.nasc.ano = 1988;
```

strcpy(estudante.nome, "José");

Lembrete

- registros podem ser atribuídos automaticamente para registros, não havendo necessidade de fazê-los campo a campo.

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

struct tipoEstudante{
      int rga;
      char nome[51];
      tipoData nasc;
};
tipoEstudante estudante1,
                    estudante2, aux;
```

```
aux = estudante1;
estudante1 = estudante2;
estudante2 = aux;
```

Problema

- Dados um inteiro positivo *n*, uma sequência de *n* nomes, telefones e datas de aniversário, e uma data no formato **dd/mm**, imprima os nomes e telefones das pessoas que aniversariam nesta data. Considere que cada nome tem no máximo 50 caracteres, cada telefone é um número inteiro positivo com 8 dígitos e cada data de aniversário tem o formato **dd/mm/aaaa**.

Solução problema

```
#include <stdio.h>
#define DIM 100
#define MAX 50

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

struct tipoAgenda{
	char nome[MAX+1];
	int telefone;
	tipoData aniver;
};

int main()
{
	tipoAgenda agenda[DIM];
	tipoData data;
	int i, n;
```

/*Recebe um inteiro positivo n e n nomes, telefones e datas de aniversário, recebe uma data para consultar e mostra os nomes e telefones de todos que aniversariam naquela data*/

```
    printf(“Informe a quantidade de amigos: ”);
    scanf("%d", &n);

     /*leitura dos dados do vetor*/
    for (i = 0; i < n; i++)
    {
          scanf(" %[^\n]", agenda[i].nome);
          scanf("%d", &agenda[i].telefone);
          scanf("%d/%d/%d", &agenda[i].aniver.dia, &agenda[i].aniver.mes,
                                        &agenda[i].aniver.ano);
    }
    printf("\nInforme uma data (dd/mm): ");
    scanf("%d/%d", &data.dia, &data.mes);

    for (i = 0; i < n; i++)
          if (agenda[i].aniver.dia == data.dia && agenda[i].aniver.mes == data.mes)
                printf("%s, %d", agenda[i].nome , agenda[i].telefone);

    return 0;
}
```

Funções para manipular Cadeia de caracteres (string)

```
#include <string.h>  // manipulação de strings

char   str1[10] = "formiga";
char   str2[10];
int   tamanho;

// calculando o tamanho de uma cadeia
tamanho = strlen(str1);  // 7
```

Funções para manipular Cadeia de caracteres (string)

```
#include <string.h>  // manipulação de strings

char   str1[10] = "formiga";
char   str2[10];

// Copiando uma cadeia para outra
strcpy(destino, origem);
strcpy(str2, str1);   // str2 receberá "formiga"
str2 = str1;  // ERRO
```

```
#include <string.h>  // manipulação de strings

char   str1[10] = "formiga", str2[10] = "gato";

// Comparando duas strings lexicograficamente
```

strcmp(str1, str2)

retorna

- 0 - se str1 = str2
- < 0 - se str1 < str2 na ordem alfabética
- > 0 - se str1 > str2 na ordem alfabética

```
#include <string.h>  // manipulação de strings

char   str1[10] = "formiga", str2[10] = "gato";

// Comparando duas strings lexicograficamente
    if(  strcmp(str1, str2 ) == 0) // (!strcmp(str1, str2))
        printf("As cadeias sao iguais");
    else if(strcmp(str1, str2 ) < 0))
        printf("%s vem antes de %s", str1, str2);
    else
        printf("%s vem após %s", str1, str2);
```