AULA 16

# VETORES, MATRIZES E REGISTROS

Nesta aula, vamos trabalhar com uma extensão natural do uso de registros, declarando e usando variáveis compostas homogêneas de registros como, por exemplo, vetores de registros ou matrizes de registros. Por outro lado, estudaremos também registros contendo variáveis compostas homogêneas como campos. Veremos que combinações dessas declarações também podem ser usadas de modo a representar e organizar os dados na memória para solução de problemas. Além disso, veremos ainda registros cujos campos podem ser variáveis não somente de tipos primitivos, de tipos definidos pelo usuário, ou ainda variáveis compostas homogêneas, mas também de variáveis compostas heterogêneas ou registros. O conjunto dessas combinações de variáveis que acabamos de mencionar fornece ao(à) programador(a) uma liberdade e flexibilidade na declaração de qualquer estrutura para armazenamento de informações que lhe seja necessária na solução de um problema computacional, especialmente daqueles problemas mais complexos.

Esta aula é baseada nas referências [10, 6].

## 16.1 Variáveis compostas homogêneas de registros

Como vimos nas aulas 12 e 14, podemos declarar variáveis compostas homogêneas a partir de qualquer tipo básico ou ainda de um tipo definido pelo(a) programador(a). Ou seja, podemos declarar, por exemplo, um vetor do tipo inteiro, uma matriz do tipo ponto flutuante, uma variável composta homogênea de $k$ dimensões do tipo caracter e etc. A partir de agora, poderemos também declarar variáveis compostas homogêneas, de quaisquer dimensões, de registros. Por exemplo, podemos declarar um vetor com identificador `cronometro` como mostrado a seguir:

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

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

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

O vetor `cronometro` declarado anteriormente contém 10 compartimentos de memória, sendo que cada um deles é do tipo registro. Cada registro contém, por sua vez, os campos `horas`, `minutos` e `segundos`, do tipo inteiro. Já a matriz com identificador `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`. Essas duas variáveis compostas homogêneas também poderiam ter sido declaradas em conjunto, numa mesma sentença de declaração, como apresentado a seguir:

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

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

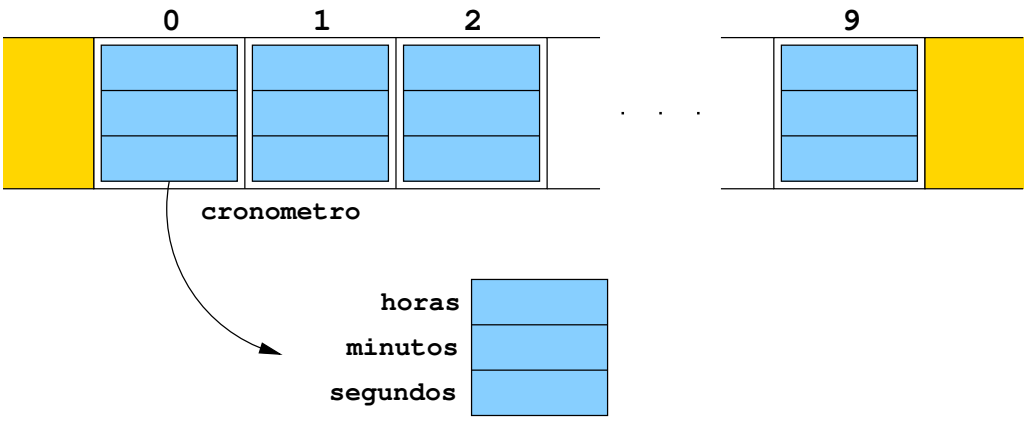


Figura 16.1: Efeito da declaração do vetor `cronometro` na memória.

Atribuições de valores do tipo inteiro aos campos do registro no primeiro compartimento deste vetor `cronometro` podem ser realizadas usando o identificador do vetor, o índice 0 (zero) envolvido por colchetes, o símbolo seletor de um campo `.` e o identificador do campo, como ilustrado nas atribuições abaixo:

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

Além disso, como fizemos na aula 15, podemos fazer a atribuição direta de registros para registros declarados da mesma maneira. Assim, por exemplo, se declaramos as variáveis `cronometro` e `aux` como abaixo:

```
struct {
   int horas;
   int minutos;
   int segundos;
} 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;
```

É importante observar novamente que todos os campos de um registro são atualizados automaticamente quando da atribuição de um registro a outro registro, não havendo a necessidade da atualização campo a campo. Ou seja, 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;
```

As duas formas de atribuição acima estão corretas, apesar da primeira forma ser muito mais prática e direta.

Nesse contexto, considere o seguinte problema:

> Dado um número inteiro $n$, com $1 \leqslant n \leqslant 100$, 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 16.1 soluciona o problema acima usando o método de ordenação das trocas sucessivas ou método da bolha.

Programa 16.1: Um programa usando um vetor de registros.

```
#include <stdio.h>

/* Recebe um inteiro n, 1 ⩽ n ⩽ 100, e n medidas de tempo
   hh:mm:ss, e mostra esses tempos em ordem crescente */
int main(void)
{
   int i, j, n;
   struct {
      int hh;
      int mm;
      int ss;
   } cron[100], aux;

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

## 16.2 Registros contendo variáveis compostas homogêneas

Na aula 15 definimos registros que continham campos de tipos básicos de dados ou, no máximo, de tipos definidos pelo(a) programador(a). Na seção 16.1, estudamos vetores não mais de tipos básicos de dados, mas de registros, isto é, vetores contendo registros. Essa idéia pode ser estendida para matrizes ou ainda para variáveis compostas homogêneas de qualquer dimensão. Por outro lado, podemos também declarar registros que contêm variáveis compostas homogêneas como campos. Um exemplo bastante comum é a declaração de um vetor de caracteres, ou uma cadeia de caracteres, dentro de um registro, isto é, como um campo deste registro:

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

A declaração do registro `mes` permite o armazenamento de um valor do tipo inteiro no campo `dias`, que pode representar, por exemplo, a quantidade de dias de um mês, e de três valores do tipo caracter no campo vetor `nome`, que podem representar os três primeiros caracteres do nome de um mês do ano. Assim, se declaramos as variáveis `mes` e `aux` como a seguir:

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

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

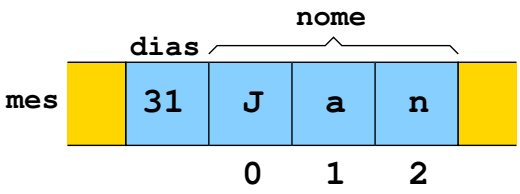


Figura 16.2: Variável `mes` na memória.

É importante salientar mais uma vez que 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:

```
aux = mes;
```

Suponha agora que temos o seguinte 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.

Uma solução para esse problema é apresentada no programa 16.2.

Programa 16.2: Exemplo do uso de um vetor como campo de um registro.

```
#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 */
int main(void)
{
   int tempo1, tempo2;
   struct {
      int horas;
      int minutos;
      int segundos;
      char descricao[51];
   } 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;
}
```

## 16.3 Registros contendo registros

É importante observar que a declaração de um registro pode conter um outro registro como um campo, em seu interior. Ou seja, uma variável composta heterogênea pode conter campos de tipos básicos, iguais ou distintos, campos de tipos definidos pelo usuário, campos que são variáveis compostas homogêneas, ou ainda campos que se constituem também como variáveis compostas heterogêneas.

Como exemplo, podemos declarar uma variável do tipo registro com nome ou identificador `estudante` contendo um campo do tipo inteiro `rga`, um campo do tipo vetor de caracteres `nome` e um campo do tipo registro `nascimento`, contendo por sua vez três campos do tipo inteiro com identificadores `dia`, `mes` e `ano`:

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

A figura 16.3 mostra o efeito da declaração da variável `estudante` na memória.

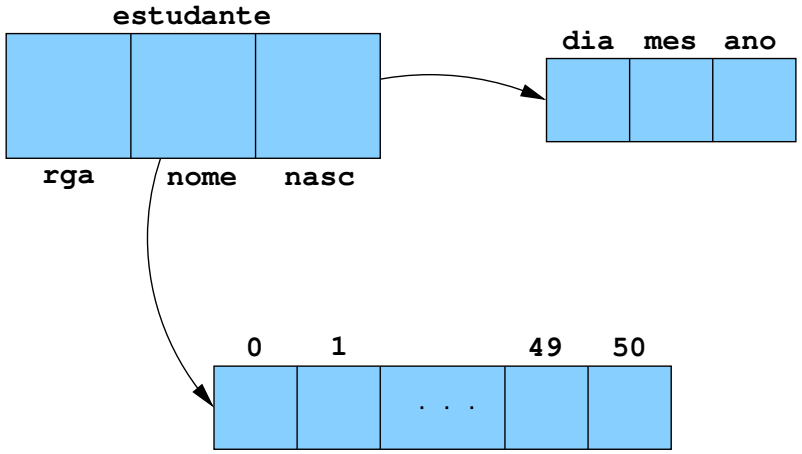


Figura 16.3: Efeitos da declaração do registro `estudante` na memória.

Observe que a variável `estudante` é um registro que mistura campos de um tipo básico – o campo `rga` que é do tipo inteiro –, um campo que é um vetor de um tipo básico – o campo `nome` do tipo vetor de caracteres – e um campo do tipo registro – o campo `nascimento` do tipo registro, contendo, por sua vez, três campos do tipo inteiro: os campos `dia`, `mes` e `ano`. 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;
```

Suponha finalmente que temos o seguinte 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`**.

Uma solução para esse problema é apresentada no programa 16.3.

Podemos destacar novamente, assim como fizemos na aula 15, que registros podem ser atribuídos automaticamente para registros, não havendo necessidade de fazê-los campo a campo. Por exemplo, se temos declarados os registros:

```
struct {
   int rga;
   char nome[51];
   struct {
      int dia;
      int mes;
      int ano;
   } nascimento;
} estudante1, estudante2, aux;
```

então, as atribuições a seguir são perfeitamente válidas e realizam corretamente a troca de conteúdos dos registros **`estudante1`** e **`estudante2`**:

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

Programa 16.3: Um exemplo de uso de registros contendo registros.

```
#include <stdio.h>

#define DIM 100
#define MAX 50

/* Recebe um inteiro positivo n e n nomes, telefones e datas de
   aniversário, recebe uma data de consulta e mostra os nomes
   e telefones das pessoas que aniversariam nesta data */
int main(void)
{
   int i, n;
   struct {
      char nome[MAX+1];
      int telefone;
      struct {
         int dia;
         int mes;
         int ano;
      } aniver;
   } agenda[DIM];
   struct {
      int dia;
      int mes;
   } data;

   printf("Informe a quantidade de amigos: ");
   scanf("%d", &n);
   for (i = 0; i < n; i++) {
      printf("\nAmigo(a): %3d\n", i+1);
      printf("   Nome       : ");
      scanf(" %[^\n]", agenda[i].nome);
      printf("   Telefone   : ");
      scanf("%d", &agenda[i].telefone);
      printf("   Aniversário: ");
      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("%-50s %8d\n", agenda[i].nome, agenda[i].telefone);

   return 0;
}
```

## Exercícios

16.1 Dados um número inteiro $n$, com $1 \leqslant n \leqslant 100$, $n$ datas no formato **dd/mm/aaaa**, e uma data de referência $D$ no mesmo formato, verifique qual das $n$ datas fornecidas é mais próxima à data $D$.

Podemos usar a fórmula do exercício 15.3 para solucionar esse exercício mais facilmente.

Programa 16.4: Solução do exercício 16.1.

```
#include <stdio.h>

#define MAX 100

/* Recebe um inteiro n, 1 <= n <= 100, n datas e uma data de referência,
   e verifica qual das n datas é mais próxima da data de referência */
int main(void)
{
   int dif[MAX], menor;
   struct {
      int dia;
      int mes;
      int ano;
   } D, data[MAX];

   printf("Informe uma data de referência (dd/mm/aa): ");
   scanf("%d/%d/%d", &D.dia, &D.mes, &D.ano);
   printf("Informe a quantidade de datas: ");
   scanf("%d", &n);
   for (i = 0; i < n; i++) {
      printf("[%03d] Informe uma data (dd/mm/aa): ", i+1);
      scanf("%d/%d/%d", &data[i].dia, &data[i].mes, &data[i].ano);
   }
   if (d.mes <= 2)
      N1 = (1461*(D.ano-1))/4 + ((153*D.mes+13)/5) + D.dia;
   else
      N1 = (1461*D.ano)/4 + ((153*D.mes+1)/5) + D.dia;
   for (i = 0; i < n; i++) {
      if (data[i].mes <= 2)
         N2 = (1461*(data[i].ano-1))/4+((153*data[i].mes+13)/5)+data[i].dia;
      else
         N2 = (1461*data[i].ano)/4 + ((153*data[i].mes+1)/5) + data[i].dia;
      if (N1 >= N2)
         dif[i] = N1 - N2;
      else
         dif[i] = N2 - N1;
   }
   menor = 0;
   for (i = 1; i < n; i++)
      if (dif[i] < dif[menor])
         menor = i;
   printf("Data mais próxima de %2d/%2d/%2d é %2d/%2d/%d\n",
      D.dia, D.mes, D.ano, data[menor].dia, data[menor].mes, data[menor].ano);

   return 0;
}
```

16.2 Dadas três fichas de produtos de um supermercado, contendo as informações de seu código (um número inteiro positivo, sua descrição (uma cadeia de caracteres com até 50 caracteres) e seu preço unitário (um número real), ordená-las em ordem alfabética de seus nomes.

Este exercício é semelhante ao exercício 5.10.

16.3 Dados um número inteiro $n$, com $1 \leqslant n \leqslant 100$, e $n$ fichas de doadores de um banco de sangue, contendo um número inteiro positivo que representa o código do doador, uma cadeia de caracteres com até 50 caracteres que representa o nome do doador, uma cadeia com até 2 caracteres que representa o tipo sanguíneo do doador (A, B, AB ou O) e um caractere que representa o fator Rh (**+** ou **−**), escreva um programa que lista os doadores do banco das seguintes formas: (*i*) em ordem crescente de códigos de doadores; (*ii*) em ordem alfabética de nomes de doadores e (*iii*) em ordem alfabética de tipos sanguíneos e fator Rh.

16.4 Suponha que em um determinado galpão estejam armazenados os materiais de construção de uma loja que vende tais materiais. Este galpão é quadrado e mede $20 \times 20 = 400\text{m}^2$ e a cada $2 \times 2 = 4\text{m}^2$ há uma certa quantidade de um material armazenado. O encarregado do setor tem uma tabela de 10 linhas por 10 colunas, representando o galpão, contendo, em cada célula, o código do material, sua descrição e sua quantidade. O código do material é um número inteiro positivo, a descrição o material contém no máximo 20 caracteres e a quantidade do material é um número real.

Escreva um programa que receba as informações armazenadas na tabela do encarregado e liste cada material e a sua quantidade disponível no galpão. Observe que um mesmo material pode encontrar-se em mais que um local no galpão.

16.5 Escreva um programa que receba o nome, o telefone e a data de nascimento de $n$ pessoas, com $1 \leqslant n \leqslant 100$, e implemente uma agenda telefônica com duas listagens possíveis: (*i*) uma lista dos nomes, telefones e datas de aniversário das pessoas em ordem alfabética de nomes e (*ii*) uma lista dos nomes, telefones e datas de aniversário das pessoas em ordem crescente de datas de aniversários das pessoas. Considere que cada nome tenha no máximo 50 caracteres, o telefone é composto por um númro inteiro positivo com até 3 dígitos representando o DDD e mais um número inteiro de 8 dígitos representando o número do telefone, e a data de nascimento é fornecida no formato **`dd/mm/aaaa`**.