AULA 28

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 em memória e solucionar problemas.

28.1 Vetores de registros

Como vimos nas aulas 19 e 24, 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 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, 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 28.1.

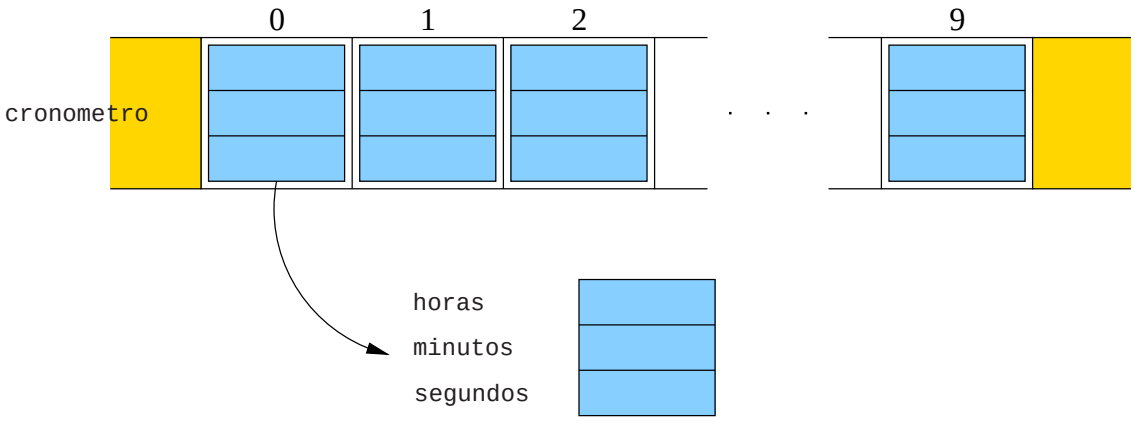

Figura 28.1: Efeito da declaração do vetor `cronometro` na memória.

Atribuições de valores do tipo inteiro aos campos do registro do primeiro compartimento deste vetor `cronometro` podem ser feitas da seguinte forma:

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

Além disso, como já mencionamos na aula 27, podemos fazer a atribuição direta de registros para registros. 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 acima estão corretas, apesar da primeira forma ser 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 28.1 soluciona o problema acima usando o método de ordenação das trocas sucessivas ou método da bolha.

Programa 28.1: Um programa usando um vetor de registros.

```
#include <stdio.h>
int main(void)
{
  int i, j, n;
  struct {
    int horas;
    int minutos;
    int segundos;
  } 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].horas, &cron[i].minutos, &cron[i].segundos);
  }
  for (i = n-1; i > 0; i--)
    for (j = 0; j < i; j++)
      if (cron[j].horas > cron[j+1].horas) {
        aux = cron[j];
        cron[j] = cron[j+1];
        cronometro[j+1] = aux;
      }
      else if (cron[j].horas == cron[j+1].horas)
        if (cron[j].minutos > cron[j+1].minutos) {
          aux = cron[j];
          cron[j] = cron[j+1];
          cron[j+1] = aux;
        }
        else if (cron[j].minutos == cron[j+1].minutos)
          if (cron[j].segundos > cron[j+1].segundos) {
            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].horas, cron[i].minutos, cron[i].segundos);
  return 0;
}
```

28.2 Registros contendo variáveis compostas

Na aula 27 definimos registros que continham campos de tipos básicos ou de tipos definidos pelo(a) programador(a). Na seção 28.1, estudamos vetores não mais de um tipo básico, mas de registros, isto é, vetores contendo registros. Podemos também declarar registros que contêm variáveis compostas como campos. Um 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[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 28.2.

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;
```

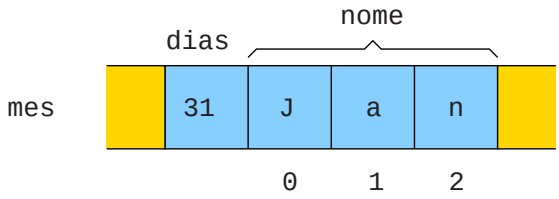

Figura 28.2: Variável `mes` na memória.

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 28.2.

Programa 28.2: Exemplo do uso de um vetor como campo de um registro.

```
#include <stdio.h>
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;
}
```

Exercícios

28.1 Dadas n datas, com $1 \leqslant n \leqslant 100$, e uma data de referência d, verifique qual das n datas é mais próxima à data d.

Podemos usar a fórmula do exercício 27.3 para solucionar esse exercício mais facilmente.

28.2 Dadas três fichas de produtos de um supermercado, contendo as informações de seu código, sua descrição com até 50 caracteres e seu preço unitário, ordená-las em ordem alfabética de seus nomes.

28.3 Dados um número inteiro $n > 1$ e mais n fichas de doadores de um banco de sangue, contendo o código do doador, seu nome, seu tipo sangüíneo e seu fator Rhesus, 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 sangüíneos e fatores Rhesus.

Programa 28.3: Solução do exercício 28.1.

```
#include <stdio.h>
#define MAX 100
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);
  scanf("%d", &n);
  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;
}
```