AULA 29

REGISTROS COM REGISTROS

Assim como na aula 28, a aula de hoje também trata da declaração de registros mais complexos, mas agora seus campos podem ser variáveis não somente de tipos básicos, de tipos definidos pelo usuário, ou ainda variáveis compostas homogêneas, mas também de variáveis compostas heterogêneas ou registros.

Uma variável que é um registro e que, por sua vez, contém como campos um ou mais registros, 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.

29.1 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 um exemplo, podemos declarar uma variável do tipo registro com identificador `estudante` contendo um campo do tipo inteiro `rga`, um campo do tipo vetor de caracteres `nome`, com 51 posições e um campo do tipo registro `nascimento`, contendo por sua vez três campos do tipo inteiro com identificadores `dia`, `mes` e `ano`. Ou seja, o registro `estudante` pode ser declarado como a seguir:

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

A figura 29.1 mostra o efeito da declaração da variável `estudante` na memória.

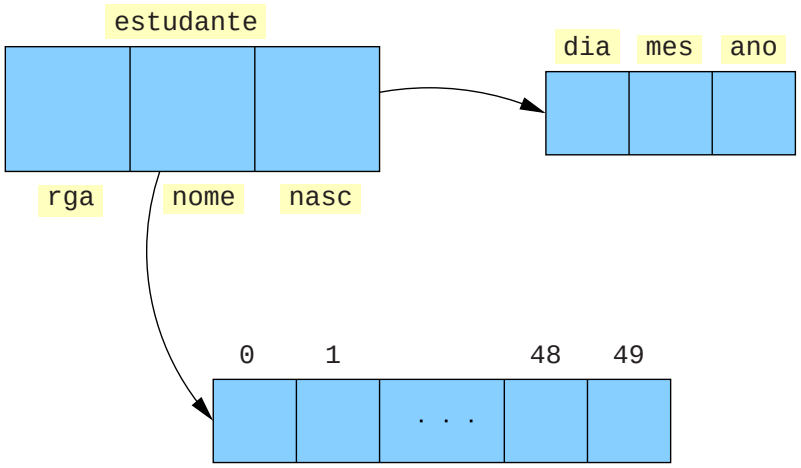

Figura 29.1: Efeitos da declaração do registro `estudante` na memória.

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;
```

29.2 Exemplo

O programa 29.1 implementa uma agenda de telefones simples, ilustrando um exemplo de uso de um vetor que contém registros, onde cada registro contém também um registro como campo. A agenda contém três informações para cada amigo(a) incluído(a): nome, telefone e data de aniversário. Além de armazenar essa agenda na memória, em um vetor de registros, o programa ainda faz com que uma data seja fornecida pelo(a) usuário(a) e todas as pessoas que fazem aniversário nessa data são mostradas na saída.

Podemos destacar novamente, assim como fizemos nas aulas 27 e 28, que registros podem ser atribuídos automaticamente para registros, não havendo necessidade de fazê-los campo a campo. Por exemplo, se temos declarados os registros:

Programa 29.1: Um exemplo de uso de registros contendo registros.

```
#include <stdio.h>
int main(void)
{
  int i, n;
  struct {
    char nome[51];
    int telefone;
    struct {
      int dia;
      int mes;
      int ano;
    } aniver;
  } agenda[100];
  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);
  i = 0;
  while (i < n) {
    if (agenda[i].aniver.dia == data.dia && agenda[i].aniver.mes == data.mes)
      printf("%-50s  %8d\n", agenda[i].nome, agenda[i].telefone);
    i++;
  }
  return 0;
}
```

```
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;
```

Exercícios

29.1 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, a descrição o material contém no máximo 20 caracteres e a quantidade do material é um número de ponto flutuante.

Escreva um programa que receba as informações armazenadas na tabela do encarrerado 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.

O programa 29.2 contém uma solução para o exercício 29.1. Observe que essa solução é dada por um código mais compacto que difere de todos os códigos que já apresentamos em nossas aulas. O principal motivo dessa divergência é a necessidade de fazer com que o programa 29.2 caiba em uma única folha.

29.2 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 e telefones das pessoas em ordem alfabética de nomes e (*ii*) uma lista dos nomes e telefones das pessoas em ordem de datas de aniversários das pessoas.

Programa 29.2: Solução do exercício 29.1.

```
#include <stdio.h>
#define MAX 10
int main(void)
{
  int i, j, k, l, m;
  float soma;
  struct {
    int codigo;
    char descricao[21];
    float quant;
    int marca;
  } galpao[MAX][MAX], resumo[MAX*MAX];
  for(i = 0; i < MAX; i++)
    for (j = 0; j < MAX; j++) {
      printf("Código do material: ");
      scanf("%d", &galpao[i][j].codigo);
      printf("Descrição: ");
      scanf(" %[^\n]", galpao[i][j].descricao);
      printf("Quantidade: ");
      scanf("%f", &galpao[i][j].quant);
      galpao[i][j].marca = 0;
    }
  for (m = 0, i = 0; i < MAX; i++)
    for (j = 0; j < MAX; j++)
      if (!galpao[i][j].marca) {
        if (j < MAX - 1) { k = i; l = j + 1; }
        else { k = i + 1; l = 0; }
        soma = galpao[i][j].quant;
        for ( ; k < MAX; k++, l = 0)
          for ( ; l < MAX; l++)
            if (galpao[k][l].codigo == galpao[i][j].codigo) {
              soma = soma + galpao[k][l].quant;
              galpao[k][l].marca = 1;
            }
        resumo[m] = galpao[i][j];
        resumo[m].quant = soma;
        m++;
      }
  for (i = 0; i < m; i++) {
    printf("\nCódigo    : %d\n", resumo[i].codigo);
    printf("Descrição : %s\n", resumo[i].descricao);
    printf("Quantidade: %5.2f\n", resumo[i].quant);
  }
  return 0;
}
```