AULA 15

REGISTROS

Nas aulas 12, 13 e 14 aprendemos a trabalhar com variáveis compostas homogêneas unidimensionais e bidimensionais – ou os vetores e as matrizes –, que permitem que um programador agrupe valores de um mesmo tipo em uma única entidade lógica. Como mencionamos antes, a dimensão de uma variável composta homogênea não fica necessariamente restrita a uma ou duas. Isto é, também podemos declarar e usar variáveis compostas homogêneas com três ou mais dimensões. Importante relembrar também que, para referenciar um elemento em uma variável composta homogênea são necessários o seu identificador e um ou mais índices.

A linguagem C dispõe também de uma outra forma para agrupamento de dados, chamada variável composta heterogênea, registro ou estrutura[1]. Nelas, diferentemente do que nas variáveis compostas homogêneas, podemos armazenar sob uma mesma entidade lógica valores de tipos diferentes. Além disso, ao invés de índices ou endereços usados nas variáveis compostas homogêneas para acesso a um valor, especificamos o nome de um campo para selecionar um campo particular do registro. Os registros são os objetos de estudo desta aula, que é inspirada nas referências [6, 10].

15.1 Definição

Uma **variável composta heterogênea** ou **registro** é uma estrutura onde podemos armazenar valores de tipos diferentes sob uma mesma entidade lógica. Cada um desses possíveis valores é armazenado em um compartimento do registro denominado **campo do registro**, ou simplesmente **campo**. Um registro é composto pelo seu identificador e pelos seus campos.

Suponha que queremos trabalhar com um agrupamento de valores que representam uma determinada mercadoria de uma loja, cujas informações relevantes são o código do produto e seu valor. Na linguagem C, podemos então declarar um registro com identificador `produto` da seguinte forma:

```
struct {
   int codigo;
   int quant;
   float valor;
} produto;
```

[1] *struct* e *member (of a structure)* são jargões comuns na linguagem C. A tradução literal dessas palavras pode nos confundir com outros termos já usados durante o curso. Por isso, escolhemos 'registro' e 'campo (do registro)' como traduções para o português.

A figura 15.1 mostra a disposição do registro `produto` na memória do computador.

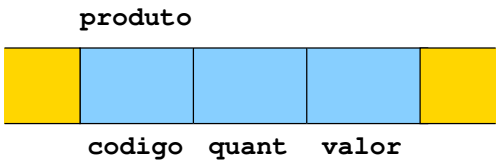

Figura 15.1: Representação do registro `produto` na memória.

Na linguagem C, a declaração de um registro sempre inicia com a palavra-chave `struct`. Logo em seguida, o delimitador de bloco `{` deve ocorrer. Depois disso, um bloco de declarações é iniciado. Nesse bloco, podemos declarar os campos do registro, linha após linha. Cada linha deve conter o tipo, de um dos tipos primitivos da linguagem C ou de um tipo criado pelo(a) programador(a), e mais o identificador do campo do registro. Finalizamos então as declarações dos campos do registro o delimitador de bloco `}`. Depois disso, realizamos de fato a declaração do registro, digitando o seu identificador. A variável do tipo registro com identificador `produto` declarada acima contém três campos, dois do tipo inteiro com identificadores `codigo` e `quant`, e outro do tipo ponto flutuante `valor`.

O formato geral de declaração de um registro é apresentado abaixo:

```
struct {
   ⋮
   bloco de declarações
   ⋮
} identificador;
```

Podemos declarar outras variáveis do tipo registro com os mesmos campos da variável `produto`, declarada acima, da seguinte forma:

```
struct {
   int codigo;
   int quant;
   float valor;
} produto, estoque, baixa;
```

Uma maneira alternativa de declaração dos campos de um registro é declarar campos de um mesmo tipo em uma única linha de código, seperando-os com vírgulas. Por exemplo, na linguagem C, a instrução a seguir:

```
struct {
   char sala, turma;
   int horas_inicio, minutos_inicio, horas_fim, minutos_fim;
   float largura, comprimento;
} aula;
```

declara um registro com identificador `aula` com seis campos, dois campos do tipo `char`, quatro outros campos do tipo `int` e dois campos do tipo `float`. Apesar dessa declaração estar correta e de economizar algumas linhas de código, sempre optamos pela declaração dos campos separadamente, linha a linha, para que os campos fiquem bem destacados e, assim, facilitem sua identificação rápida no código. Dessa forma, em geral optamos pela seguinte declaração do registro `aula`:

```
struct {
   char sala;
   char turma;
   int horas_inicio;
   int minutos_inicio;
   int horas_fim;
   int minutos_fim;
   float largura;
   float comprimento;
} aula;
```

Diferentemente da atribuição de um valor a uma variável ou a um compartimento de uma variável composta homogênea, a atribuição de um valor a um campo de uma variável do tipo registro é realizada através do acesso a esse campo, especificando o identificador do registro, um ponto e o identificador do campo.

Por exemplo, para atribuir os valores $12, 5$ e 34.5 aos campos `codigo`, `quant` e `valor`, respectivamente, da variável do tipo registro `produto` declarada anteriormente, devemos fazer como abaixo:

```
produto.codigo = 12;
produto.quant = 5;
produto.valor = 34.5;
```

Observe acima que, quando referenciamos um campo de uma variável do tipo registro, não são permitidos espaços entre o identificador do registro, o operador 'ponto' e o identificador do campo.

Também podemos usar o valor de um campo de um registro em quaisquer expressões. Por exemplo, a expressão relacional abaixo é correta:

```
if (produto.valor < 150.0)
   printf("Comprar produto\n");
else
   printf("Acima do preço de mercado!\n");
```

Declarações de registros diferentes podem conter campos com mesmo identificador. Por exemplo,

```
struct {
   char tipo;
   char fatorRH;
   int idade;
   float altura;
} coleta;

struct {
   char codigo;
   int tipo;
   int idade;
} certidao;
```

são declarações válidas na linguagem C. O acesso aos campos dessas variáveis se diferencia justamente pelo identificador dos registros. Isto é, a partir das declarações dos registros `coleta` e `certidao`, as atribuições abaixo estão corretas:

```
coleta.tipo = 'O';
certidao.tipo = 0;
coleta.idade = 29;
certidao.idade = coleta.idade + 2;
```

15.2 Declaração e inicialização simultâneas

Declaração e inicialização simultâneas também são válidas com registros. As regras são idênticas às dos vetores. Por exemplo, o registro `produto` que declaramos acima pode ter sua inicialização realizada no momento de sua declaração como segue:

```
struct {
   int codigo;
   int quant;
   float valor;
} produto = {1, 5, 34.5};
```

Neste caso, o campo `codigo` é inicializado com o valor `1`, o campo `quant` é inicializado com o valor `5` e o campo `valor` com `34.5`. Como mencionado, as regras de declaração e inicialização simultâneas para registros são equivalentes àquelas para vetores. Assim,

```
   struct {
      int codigo;
      int quant;
      float valor;
   } produto = {0};
```

fará a inicialização dos campos `codigo` e `quant` com `0` e do campo `valor` com `0.0`.

15.3 Operações sobre registros

Variáveis compostas heterogêneas possuem uma característica importante, que não é observada em variáveis compostas homogêneas.

Lembre-se que quando queremos copiar o conteúdo completo de todos os compartimentos de uma variável composta homogênea para os compartimentos respectivos de outra variável composta homogênea, é necessário realizar a cópia elemento a elemento, escrevendo uma ou mais estruturas de repetição para que essa cópia seja efetuada com sucesso. Isto é, se `A` e `B` são, por exemplo, vetores de um mesmo tipo de dados e mesma dimensão, é errado tentar fazer uma atribuição como abaixo:

```
A = B;
```

para copiar os valores do vetor `B` no vetor `A`. O compilador da linguagem C deve acusar um erro como esse. O correto então, neste caso, é fazer a cópia elemento a elemento de uma variável para outra. Supondo que n é a dimensão desses vetores, uma forma correta de realizar essa cópia é mostrada a seguir:

```
for (i = 0; i < n; i++)
   A[i] = B[i];
```

Por outro lado, quando tratamos de registros, podemos fazer uma atribuição direta e realizar a cópia de todos os seus campos nessa única atribuição. O trecho de código a seguir mostra um exemplo com a declaração de dois registros com mesmos campos, a atribuição de valores ao primeiro registro e uma cópia completa de todos os campos do primeiro registro para o segundo:

```
⋮
struct {
   char tipo;
   int codigo;
   int quant;
   float valor;
} mercadoria1, mercadoria2;
⋮
mercadoria1.tipo = 'A';
mercadoria1.codigo = 10029;
mercadoria1.quant = 62;
mercadoria1.valor = 10.32 * TAXA + 0.53;
⋮
mercadoria2 = mercadoria1;
⋮
```

15.4 Exemplo

O programa 15.1 a seguir recebe um horário no formato de horas, minutos e segundos, com as horas no intervalo de 0 a 23, e atualiza este horário em um segundo.

Programa 15.1: Atualiza o horário em 1 segundo

```
#include <stdio.h>

/* Recebe um horário no formato hh:mm:ss e o atualiza
   em 1 segundo, mostrando o novo horário na saída */
int main(void)
{
   struct {
      int hh;
      int mm;
      int ss;
   } agora, prox;

   printf("Informe o horário atual (hh:mm:ss): ");
   scanf("%d:%d:%d", &agora.hh, &agora.mm, &agora.ss);

   prox = agora;

   prox.ss = prox.ss + 1;
   if (prox.ss == 60) {
      prox.ss = 0;
      prox.mm = prox.mm + 1;
      if (prox.mm == 60) {
         prox.mm = 0;
         prox.hh = prox.hh + 1;
         if (prox.hh == 24)
            prox.hh = 0;
      }
   }

   printf("Próximo horário é %d:%d:%d\n", prox.hh, prox.mm, prox.ss);

   return 0;
}
```

Obs: nos slides comentaremos outra maneira de declarar uma variável registro (externamente à função main)

Exercícios

15.1 Dada uma data no formato **`dd/mm/aaaa`**, escreva um programa que mostre a próxima data, isto é, a data que representa o dia seguinte à data fornecida.

Exemplo:

> Se a data fornecida é **`30/06/2011`** a saída deve ser **`01/07/2011`**.

Importante! Não esqueça dos anos bissextos. Lembre-se que um ano é bissexto se é divisível por 400 ou, em caso negativo, se é divisível por 4 mas não por 100.

Programa 15.2: Solução do exercício 15.1.

```
#include <stdio.h>

/* Recebe uma data no formato dd:mm:aaaa e a atua-
   liza em 1 dia, mostrando a nova data na saída */
int main(void)
{
   struct {
      int dia;
      int mes;
      int ano;
   } data, prox;

   printf("Informe uma data (dd/mm/aa): ");
   scanf("%d/%d/%d", &data.dia, &data.mes, &data.ano);

   prox = data;
   prox.dia++;

   if (prox.dia > 31 ||
       (prox.dia == 31 && (prox.mes == 4 || prox.mes == 6 ||
                           prox.mes == 9 || prox.mes == 11)) ||
      (prox.dia == 30 && prox.mes == 2) ||
      (prox.dia == 29 && prox.mes == 2 && (prox.ano % 400 != 0 &&
                              (prox.ano % 100 == 0 || prox.ano % 4 != 0)))) {
      prox.dia = 1;
      prox.mes++;
      if (prox.mes > 12) {
         prox.mes = 1;
         prox.ano++;
      }
   }

   printf("%02d/%02d/%02d\n", prox.dia, prox.mes, prox.ano);

   return 0;
}
```

Obs: nos slides comentaremos outra maneira de declarar uma variável registro (externamente à função main)

15.2 Dados dois horários de um mesmo dia, expressos no formato **`hh:mm:ss`**, calcule o tempo decorrido entre estes dois horários, apresentando o resultado no mesmo formato **`hh:mm:ss`**.

Exemplo:

> Se os dois horários fornecidos são **`07:13:22`** e **`13:05:56`**, a resposta tem de ser **`05:52:34`**.

15.3 Dadas duas datas no formato **`dd/mm/aaaa`**, calcule o número de dias decorridos entre estas duas datas.

Exemplo:

> Se as duas datas fornecidas são **`01/03/2007`** e **`23/09/2001`**, a resposta deve ser 1985.

Uma maneira provavelmente mais simples de computar essa diferença é usar a fórmula 15.1 para calcular um número de dias N baseado em uma data:

$$N = \left\lfloor \frac{1461 \times f(\text{ano}, \text{mês})}{4} \right\rfloor + \left\lfloor \frac{153 \times g(\text{mês})}{5} \right\rfloor + \text{dia} \tag{15.1}$$

onde

$$f(\text{ano}, \text{mês}) = \begin{cases} \text{ano} - 1, & \text{se mês } \leqslant 2\,, \\ \text{ano}, & \text{caso contrário} \end{cases}$$

e

$$g(\text{mês}) = \begin{cases} \text{mês} + 13, & \text{se mês } \leqslant 2\,, \\ \text{mês} + 1, & \text{caso contrário}\,. \end{cases}$$

Lembre-se ainda que o **piso de um número real** x, denotado por $\lfloor x \rfloor$, é o maior número inteiro menor ou igual a x. Ou seja, $\lfloor 5.3 \rfloor = 5$, $\lfloor 12.999 \rfloor = 12$ e $\lfloor 2 \rfloor = 2$.

Podemos calcular o valor N_1 para a primeira data informada, o valor N_2 para a segunda data informada e a diferença $|N_2 - N_1|$ é o número de dias decorridos entre estas duas datas informadas.

15.4 Seja N computado como na equação 15.1. Então, o valor

$$D = (N - 621049) \bmod 7$$

é um número entre 0 e 6 que representa os dias da semana, de domingo a sábado. Por exemplo, para a data de 21/06/2007 temos

$$\begin{aligned} N &= \left\lfloor \frac{1461 \times f(2007, 6)}{4} \right\rfloor + \left\lfloor \frac{153 \times g(6)}{5} \right\rfloor + 21 \\ &= \left\lfloor \frac{1461 \times 2007}{4} \right\rfloor + \left\lfloor \frac{153 \times 7}{5} \right\rfloor + 21 \\ &= 733056 + 214 + 21 \\ &= 733291 \end{aligned}$$

e então

$$\begin{aligned} D &= (733291 - 621049) \bmod 7 \\ &= 112242 \bmod 7 \\ &= 4\,. \end{aligned}$$

Dada uma data fornecida pelo usuário no formato **`dd/mm/aaaa`**, determine o dia da semana para esta data.