Agregados Heterogêneos: Registros

Notas de Aula

Prof. Francisco Rapchan
www.geocities.com/chicorapchan
rapchan@terra.com.br

Como já vimos, podemos usar estruturas como Vetores ou Matrizes para compor agregados (grupos de dados) que sejam todos do mesmo tipo. Por exemplo: um Vetor de inteiros ou uma Matriz de caracteres. Dizemos então que Vetores e Matrizes são agregados homogêneos porque os seus elementos são todos do mesmo tipo.

Mas há outras formas de agregar dados. Podemos ter um agregado heterogêneo. Neste tipo de agregado, podemos agrupar dados que são de tipos diferentes. Em C, compomos agregados heterogêneos usando estruturas (structs). Uma estrutura permite dados heterogêneos, isto é, composta de elementos de tipos diferentes. Cada elemento do registro é denominado campo.

Para ilustrar, suponha um programa que manipula dados de alunos:

```
struct aluno {
   char nome [41]; // campo nome de 40 posições
   float nota1;    // campo nota1 do tipo real
   float nota2;    // campo nota2 do tipo real
};
```

Neste caso usamos **struct aluno {...}** para criar a estrutura (ou registro) aluno contendo 3 campos: nome, nota1 e nota2.

Podemos então criar variáveis do tipo struct aluno:

```
struct aluno a1, a2; // a1 e a2 são variáveis do tipo registro aluno
```

Exemplo 1. Faça um programa que use um registro (*struct*) para armazenar o nome e duas notas bimestrais de um aluno.

```
#include <stdio.h>
main()
{
   struct aluno {
      char nome [41];
      float nota1;
      float nota2;
   };

   struct aluno a1, a2;

   strcpy (a1.nome, "Joao da Silva");
   a1.nota1 = 7;
   a1.nota2 = 9;

   a2 = a1;

   printf ("Nome: %s ", a2.nome);
   printf ("Notas: %f %f\n", a2.nota1, a2.nota2);

}
```

Observe que preenchemos os dados do aluno 1 (a1) e depois atribuímos estes dados ao aluno 2 (a2).

Neste exemplo não precisávamos usar a variável a2 mas a usamos para demonstrar que podemos fazer atribuições entre variáveis de mesmo tipo struct.

Exemplo 2. O programa abaixo lê dados de alunos (nome e média) e depois mostra os alunos aprovados (média maior ou igual a 7).

```
#include <stdio.h>
main()
{  // Cria as estruturas (registros)

   struct aluno {
      char  nome [41];
      float media;

   };

   struct turma {
      int    quantidade;
      struct aluno lista [5];
   };

   // Cria as variáveis
   struct turma t1;
   int    c;

   // Pergunta o número de alunos
   printf ("Numero de alunos: ");
   scanf  ("%i",&t1.quantidade);

   // Preeche os dados dos alunos
   for (c=0; c < t1.quantidade;c++){
      printf ("\nAluno %i\n",c+1);
      printf ("Nome:");
      scanf  (" %40[^\n]",t1.lista[c].nome);
      printf ("Média:");
      scanf  ("%f",&t1.lista[c].media);
   }

   // Mostra os alunos aprovados
   printf ("\nLista de Aprovados:\n\n");
   for (c = 0; c < t1.quantidade; c++){
      if (t1.lista[c].media >= 7.0)
         printf ("%s\n",t1.lista[c].nome);
   }
}
```

Simulação da Execução do Programa:
(veja a ocupação da memória ao lado)

```
Número de alunos: 3

Aluno 1
Nome: Ana
Média: 7

Aluno 2
Nome: Maria
Média: 6

Aluno 3
Nome: Carla
Média: 10

Lista de Aprovados:
Ana
Carla
```

Neste programa estamos criando duas **estruturas**. Em outras linguagens as estruturas também são chamadas de registros.

Cada estrutura é formada por uma série de **campos**.

A primeira estrutura, chamada de **aluno**, possui dois campos: nome (que é uma string de 40 posições) e media (que é um número real).

A segunda estrutura é chamada de **turma** e possui também dois campos: quantidade (que é um inteiro) e lista (que é um vetor de 5 elementos do tipo aluno).

O campo quantidade irá conter o número de alunos que serão cadastrados na lista. Esse número não pode ser maior que 5 pois o vetor lista foi definido com esse tamanho (que representa o número máximo de alunos que o programa pode cadastrar).

Para manipular os dados, o programa cria a variável **t1** do tipo struct turma. Essa variável conterá os dados da turma. Atenção: não confunda a estrutura turma com a variável t1. A variável t1 ocupa um espaço na memória do computador enquanto que a estrutura turma é apenas uma definição (não é uma variável).

Abaixo mostramos a memória alocada para a variável t1 depois da execução do exemplo.

t1

quantidade: 3

lista:

0	**nome**	Ana
	média	7.0
1	**nome**	Maria
	média	6.0
2	**nome**	Carla
	média	10.0
3	**nome**	Lixo
	média	Lixo
4	**nome**	Lixo
	média	Lixo

A Definição de Novos Tipos (typedef)

Em C é possível definir novos tipos a partir dos tipos já existentes. Para isso usamos o comando **typedef**.

Exemplo:

```
typedef float Real;          // Está criando o tipo Real que é igual ao float.
typedef char Caracter;       // Está criando o tipo Caracter.
typedef float Vetor[4];      // Está criando o tipo Vetor como um array de 4 posições do tipo float.
```

Exemplo 3. O programa abaixo exemplifica o uso do typedef.

```
#include <stdio.h>
main()
{
   // Cria os tipos
   typedef char   t_nome[41];  // Cria o tipo nome (t_nome) como uma string
   typedef int    t_inteiro;   // Cria o tipo inteiro (t_inteiro)
   typedef t_nome t_lista[10]; // Cria o tipo lista de nomes

   // Cria as variáveis
   t_lista nomes;              // Cria a variável nomes do tipo t_lista
   t_inteiro c;                // Cria a variável c do tipo t_inteiro

   // Lê os dados e armazena em nomes
   for (c=0; c < 10 ; c++){
      printf ("Nome %i:",c+1);
      scanf (" %40[^\n]",nomes[c]);
   }

   // Mostra os dados em ordem inversa da leitura
   for (c=9; c >= 0 ; c--)
      printf ("%s\n",nomes[c]);

}
```

Podemos também criar tipos a partir de estruturas. Para o exemplo apresentado anteriormente teríamos:

```
...
// Cria os tipos

typedef char t_nome[41];       // Define o tipo nome como string de 40 posições
typedef int  t_inteiro;        // Define o tipo inteiro como apelido para int

typedef struct {
   t_nome nome;
   float media;
} t_aluno;                     // Define o tipo aluno: t_aluno

typedef struct {
   int quantidade;
   t_aluno lista [100];
} t_turma;                     // Define o tipo turma: t_turma

t_turma t1;                    // Cria t1 do tipo t_turma
int     c;                     // Cria c como int (não quis usar t_inteiro!)
...
```

Observe que as estruturas foram criadas **sem nome** (no exemplo anterior elas foram chamadas de aluno e turma respectivamente). Isso acontece porque estamos definindo um novo tipo e o nome das estruturas não é mais necessário.

Exemplo 4. Faça um cadastro de produtos usando as estruturas propostas abaixo. O programa deve cadastrar produtos e depois mostrar os dados cadastrados.

t_produto Produto de venda

descricao	String 40	Nome do produto
qtde	Inteiro	Quantidade de produtos no estoque
preco	Real	Preço unitário do produto

t_estoque Estoque de produtos

qtde	Inteiro	Quantidade de produtos na lista
produtos	Vetor de 100 elementos t_produto	Vetor de produtos (máximo 100 elementos)

```
#include <stdio.h>
main()
{  // Cria as estruturas (registros)
   typedef struct {
      char  descricao [41];
      int   qtde;
      float preco;
   } t_produto;

   typedef struct {
      int qtde;
      t_produto produtos [100];
   } t_estoque;

   // Cria as variáveis
   t_estoque estoque;
   int       c;

   // Pergunta o número de produtos
   printf ("Numero de produtos: ");
   scanf  ("%i",&estoque.qtde);

   // Preeche o vetor estoque de produtos
   for (c=0; c < estoque.qtde ; c++){
      printf ("\nProduto: %i\n",c+1);
      printf ("Descricao:");
      scanf  (" %40[^\n]",estoque.produtos[c].descricao);
      printf ("Quantidade no estoque:");
      scanf  ("%i",&estoque.produtos[c].qtde);
      printf ("Preco unitario:");
      scanf  ("%f",&estoque.produtos[c].preco);
   }

   // Mostra o relatório de produtos
   printf ("\n\nRelatorio do Estoque\n");
   printf ("%-40s %7s %8s\n","Descricao","QTDE","Preco");
   for (c = 0; c < estoque.qtde; c++){
      printf ("%-40s ",estoque.produtos[c].descricao);
      printf ("%7i ",estoque.produtos[c].qtde);
      printf ("%8.2f\n",estoque.produtos[c].preco);
   }
}
```

Observe a definição dos tipos **t_produto** e **t_estoque** conforme a especificação dada.

Neste programa é importante entender a diferença de significado dos campos qtde usada em t_produto e qtde usada em t_estoque.

O campo qtde usado em t_produto refere-se à quantidade de produtos no estoque.

Já o campo qtde usado em t_estoque refere-se ao número de produtos que estarão efetivamente cadastrados no vetor estoque.

Para o compilador não há nenhuma confusão de nomes porque cada um está em uma estrutura diferente (não poderíamos ter dois campos com o mesmo nome na mesma estrutura).

Observe também a formatação usada no relatório de produtos. A função printf permite uma série de ajustes de formatação:

- "%-40s " indica serão reservadas 40 posições para a string e ela será alinhada à esquerda.
- "%7s" indica que serão reservados 7 posições com alinhamento normal (à direita).
- "%8.2f" indica que o número será representado com 8 caracteres sendo 2 deles depois da vírgula.

Exemplo 5. Altere o programa anterior para que os dados no vetor estoque sejam ordenados pelo preço do produto.

Nesta solução aproveitamos grande parte do programa do exemplo anterior. Colocamos reticências (...) onde não houve alteração.

Também vamos adotar o mesmo algoritmo de ordenação usado quando tratamos de vetores:

```
for (i = 0; i < tamanho ; i++)        // Varre o vetor todo
{  for (j = 0; j < tamanho -1; j++)   // Varre até o penúltimo elemento
   {  if (v[j] > v[j+1])              // Se o atual for maior que o próximo
      {  aux = v[j];                  // Inverte as posições (swap) para colocar
         v[j] = v[j+1];               // o maior em uma posição depois
         v[j+1] = aux;                // do menor
      }
   }
}
```

Solução:

```
#include <stdio.h>
main()
{  // Cria as estruturas (registros)
   ...
   // Cria as variáveis
   ...
   int       c, i, j;

   // Pergunta o número de produtos
   ...

   // Preeche o estoque
   ...

   // Ordena o vetor estoque pelo preço do produto
   for (i = 0; i < estoque.qtde ; i++)
   { for (j = 0; j < estoque.qtde -1 ; j++)
     {  if(estoque.produtos[j].preco > estoque.produtos[j+1].preco)
        { aux                   = estoque.produtos[j];
          estoque.produtos[j]   = estoque.produtos[j+1];
          estoque.produtos[j+1] = aux;
        }
     }
   }

   // Mostra os produtos
   ...
}
```

Exemplo 6. Altere o programa anterior para que os dados no vetor estoque sejam ordenados pelo nome do produto (ordem alfabética).

Neste caso, usamos a função strcmp que serve para comparar strings (disponível na biblioteca <string.h>).

```
   // Ordena o vetor estoque pelo preço do produto
   for (i = 0; i < estoque.qtde ; i++)
   {  for (j = 0; j < estoque.qtde -1 ; j++)
      {  if (strcmp (estoque.produtos[j].descricao,
                     estoque.produtos[j+1].descricao) > 0 )
         {  aux                   = estoque.produtos[j];
            estoque.produtos[j]   = estoque.produtos[j+1];
            estoque.produtos[j+1] = aux;
         }
      }
   }
```

Exemplo 7. Faça um programa de cadastro de turma contendo alunos e suas notas. Use as estruturas propostas abaixo.

t_aluno Tipo aluno

nome	String de 40 caracteres	Nome do aluno
qtde	Inteiro	Quantidade de notas que serão armazenadas
notas	Vetor de 100 elementos reais	Vetor de notas do aluno

t_turma Tipo turma de alunos

nome	String de 20 caracteres	Nome ou código da turma
qtde	Inteiro	Quantidade de alunos na turma
alunos	Vetor de 100 elementos t_aluno	Lista de alunos da turma

```
#include <stdio.h>
main()
{  // Cria as estruturas (registros)
   typedef struct {
      char  nome [41];
      int   qtde;
      float notas [10];
   } t_aluno;

   typedef struct {
      int qtde;
      t_aluno alunos [100];
   } t_turma;

   // Cria as variáveis
   t_turma turma;
   int c, i;
   float soma, media;

   // Preenche os dados da turma
   printf ("Numero de alunos: ");
   scanf  ("%i",&turma.qtde);

   // Preeche os dados dos alunos
   for (c=0; c < turma.qtde ; c++){
      printf ("\nAluno: %i\n",c+1);
      printf ("Nome: ");
      scanf  (" %40[^\n]",turma.alunos[c].nome);
      printf ("Numero de notas: ");
      scanf  ("%i",&turma.alunos[c].qtde);
      // Lê as notas dos alunos
      for (i = 0; i < turma.alunos[c].qtde; i++){
         printf ("Nota %i: ", i+1);
         scanf  ("%f",&turma.alunos[c].notas[i]);
      }
   }

   // Mostra os alunos e as médias
   printf ("\n\nRelatorio de Alunos\n");
   printf ("%-40s %4s\n","Nome","Media");
   for (c=0; c < turma.qtde ; c++){
      printf ("%-40s",turma.alunos[c].nome);

      // Soma as notas e calcula a média
      soma = 0;
      for (i = 0; i < turma.alunos[c].qtde; i++)
         soma = soma + turma.alunos[c].notas[i];
      media = soma / turma.alunos[c].qtde;
      printf (" %5.1f\n",media);
   }
}
```

Observe a definição dos tipos **t_aluno** e **t_turma** conforme a especificação dada.

Um outro aspecto importante neste programa é o fato de haver um vetor de notas (notas [10]) que é usado dentro de um vetor de alunos (alunos [100]).

Isso pode gerar alguma confusão ao preencher os dados. Por exemplo, a linha:

```
scanf  ("%f",&turma.alunos[c].notas[i]);
```

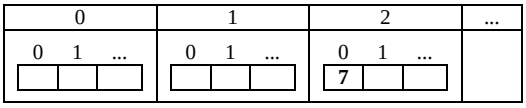
Indica dois índices:
- **[i] :** que se refere à nota do aluno que está sendo preenchida.
- **[c] :** que se refere ao aluno que está tendo as notas preenchidas.

Se estivéssemos fazendo essa atribuição um a um, poderíamos ter:

```
turma.alunos[2].notas[0] = 7;
```

Indicando tratar-se da primeira nota (posição zero) do terceiro aluno (posição dois).

Abaixo temos uma representação de memória de parte da estrutura turma contendo apenas os primeiros 3 alunos e, para cada um, as primeiras 3 notas.



Lista de Exercícios

Os exercícios abaixo estão baseados na seguinte estrutura:

t_pessoa Tipo pessoa da agenda

nome	String de 40 caracteres	Nome da pessoa
fone	String de 20 caracteres	Telefone da pessoa

t_agenda Tipo agenda

qtde	Inteiro	Quantidade de pessoas na agenda
pessoas	Vetor de 100 elementos t_pessoa	Lista de pessoas na agenda

1) Faça um programa para cadastrar o nome e o telefone de até 100 pessoas. Depois de cadastrar, mostre todas as pessoas e os telefones na forma de um relatório.

2) Altere o programa 1 para que seja pedido um nome e mostrado o telefone da pessoa (consulta).

3) Altere o programa 1 para que os dados sejam mostrados em ordem alfabética do nome (ordenação).

4) Altere o programa 1 para que permita alterar o telefone de uma pessoa cadastrada. O programa deve perguntar o nome da pessoa, mostrar o telefone antigo e perguntar o telefone novo. Ao final, mostre todas as pessoas e os telefones na forma de um relatório (alteração).

5) Altere o programa 1 para que seja perguntado um novo nome e telefone para ser incluído na agenda. Ao final, mostre todas as pessoas e os telefones na forma de um relatório (inclusão).

6) Altere o programa 1 para que seja pedido o nome de uma pessoa e ela seja retirada da agenda (exclusão). Dica: crie uma nova agenda e transfira todas as pessoas para ela (menos a que você está procurando). Depois, traga de volta os dados para a agenda antiga e mostre todas as pessoas e os telefones na forma de um relatório.

7) Altere o programa 1, incluindo um campo chamado relacionamento em t_pessoa. Nesse campo cadastre o tipo de relacionamento (namoro, família, trabalho, amigo, etc...). Depois, pergunte um relacionamento e mostre um relatório com todos os nomes e telefones de pessoas que você cadastrou com aquele relacionamento (inclusão de campo).

8) Faça as alterações necessárias no programa 1 para que possam ser cadastrados até 10 telefones para cada pessoa (alteração de campo).