

ESTRUTURA DE DADOS

ESTRUTURAS HETEROGÊNEAS

Olá!

Ao final desta aula, você será capaz de:

1. Compreender o uso das estruturas heterogêneas definidas pelo programador;
2. Definir e declarar estruturas heterogêneas localmente e globalmente;
3. Identificar que tipos de elementos podem ser membros de uma estrutura;
4. Implementar programas usando estruturas heterogêneas;
5. Construir funções usando estruturas heterogêneas.

1 Tema: As Estruturas de Dados heterogêneas chegaram para agrupar

Na disciplina de Algoritmos, estudamos as estruturas de dados homogêneas, matrizes, que eram capazes de armazenar grandes volumes de dados desde que fossem do mesmo tipo.

Fizemos vários exercícios que manipulavam tipos de dados diferentes de vários alunos, ou de vários produtos, ou de vários funcionários. Você deve estar se lembrando de quantas matrizes foram necessárias para que pudéssemos construir um programa, prejudicando, muitas vezes, a legibilidade e a manutenibilidade.

Com a introdução da estrutura de dados heterogêneas (struct), conhecida em algumas linguagens como registros, você perceberá que tudo ficará muito mais simples.

Definindo...

Podemos definir uma estrutura como sendo um conjunto de elementos, geralmente, agrupados sob uma lógica e associados por um nome.

Esses elementos podem ser variáveis simples, matrizes, outras estruturas e até funções.

Por essa definição, podemos concluir que uma estrutura pode ser formada por elementos de tipos diferentes.

Cada elemento da estrutura é chamado de membro ou campo.

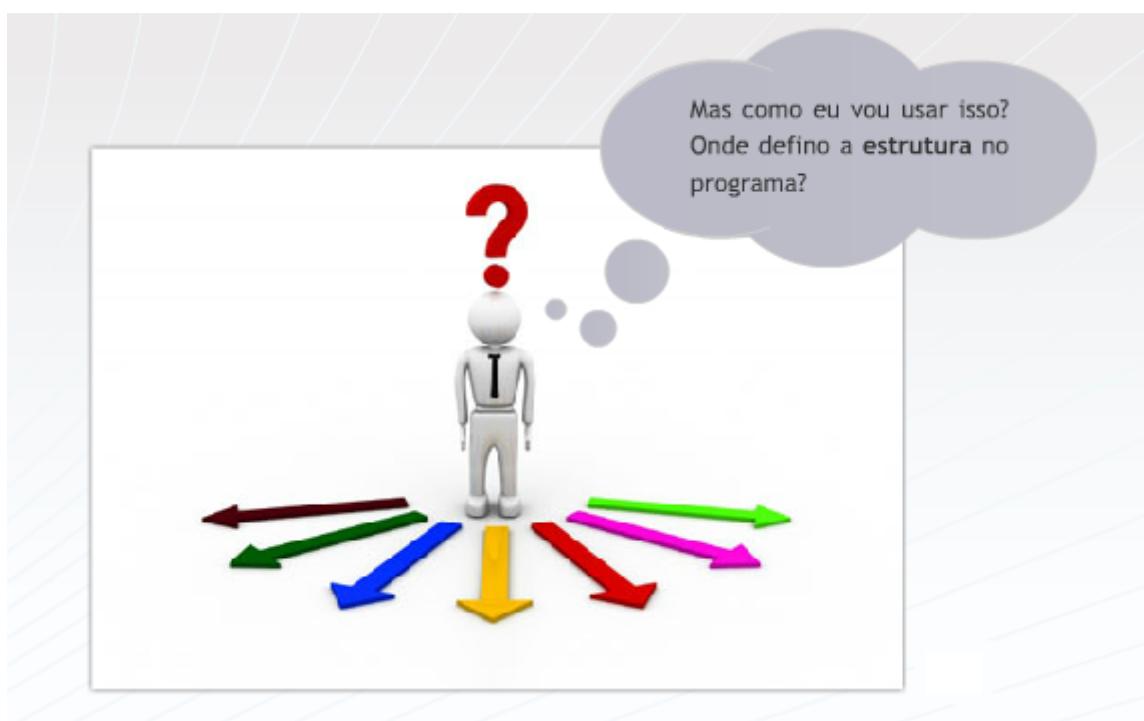
Como se define uma estrutura?

Para se definir uma estrutura, você precisa seguir a seguinte sintaxe:

struct <identificador>	struct palavra que inicia a definição da estrutura <identificador> é o nome da estrutura
{	Chave de abertura
<tipoDoMembro><identificador>;	tipoDoMembro pode ser int, float, double, char . <identificador> é o nome do membro(campo) que segue as mesmas regras das variáveis.
};	Chave de fechamento com ;

Atenção

A definição termina com um; porque é um comando. É bom ressaltar que nenhuma variável foi declarada.



Respondendo aos questionamentos...

Onde irei usar isso? Quando definimos uma estrutura, o nome dado à estrutura se torna um tipo assim como você usa int, float, double, char. Com este nome, você poderá declarar variáveis, inclusive arrays(matrizes) e, o mais importante, nem será necessário usar a palavra struct antes, só o nome que você deu. Onde se define uma estrutura? Uma estrutura pode ser definida dentro de uma função e então, dizemos que ela é urna estrutura local ou então, definida antes de todas as funções e dizemos que ela é urna estrutura global. Veja exemplos.

Global	Local
<pre>#include <iostream> using namespace std; struct exemplo1 { ... }; void ast() { } int main() { }</pre>	<pre>#include <iostream> using namespace std; void ast() { } int main() { struct exemplo2 { ... }; }</pre>

Atenção

Podemos então concluir que uma estrutura definida de forma global poderá ser usada por todas as funções.

Enquanto que uma estrutura definida localmente, só poderá ser usada na função onde foi definida.



Temos dois métodos...

1º método

A declaração é feita depois da definição. Se a definição for global, a declaração acontece em cada função.

struct nomeDaEstrutura { ... }; nomeDaEstrutura nomeDaVariável;	struct cadastro { ... }; cadastro aluno;
--	---

Temos dois métodos...

2º método

A declaração é feita junta com a definição.

struct nomeDaEstrutura { ... } nomeDaVariável;	struct cadastro { ... } aluno;
--	--

Existe uma estrutura que chamamos de estrutura anônima. Essa estrutura não tem identificador. Todas as variáveis precisarão ser declaradas na definição. Observe a sintaxe dessa estrutura.

struct { <tipoDoMembro><identificador>; <tipoDoMembro><identificador>; } variável1, variável2;	struct palavra que inicia a definição da estrutura Chave de abertura tipoDoMembro pode ser int , float , double , char . <identificador> é o nome do membro(campo) que segue as mesmas regras das variáveis. Chave de fechamento e declaração de todas as variáveis, obrigatoriamente aqui, com ;
--	--

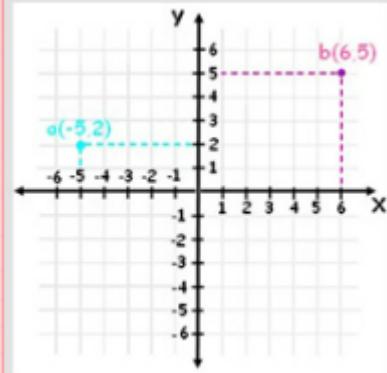
Atenção

Alguns autores chamam de identificador o nome que damos à estrutura. Outros, chama de gabarito ou, simplesmente, nome da estrutura.

Sendo assim, para facilitar o entendimento, passaremos a usar nomeDaEstrutura.

Agora que a variável foi declarada, o compilador aloca na Memória Principal todos os membros da estrutura.

Nosso primeiro exemplo será bem matemático, tendo em vista que todos se recordam das coordenadas cartesianas. Escolhemos o primeiro método.



```
struct coordenadas
{
    int x, y;
};

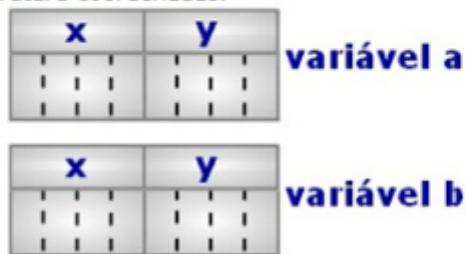
int main()
{
    coordenadas a,b;
```

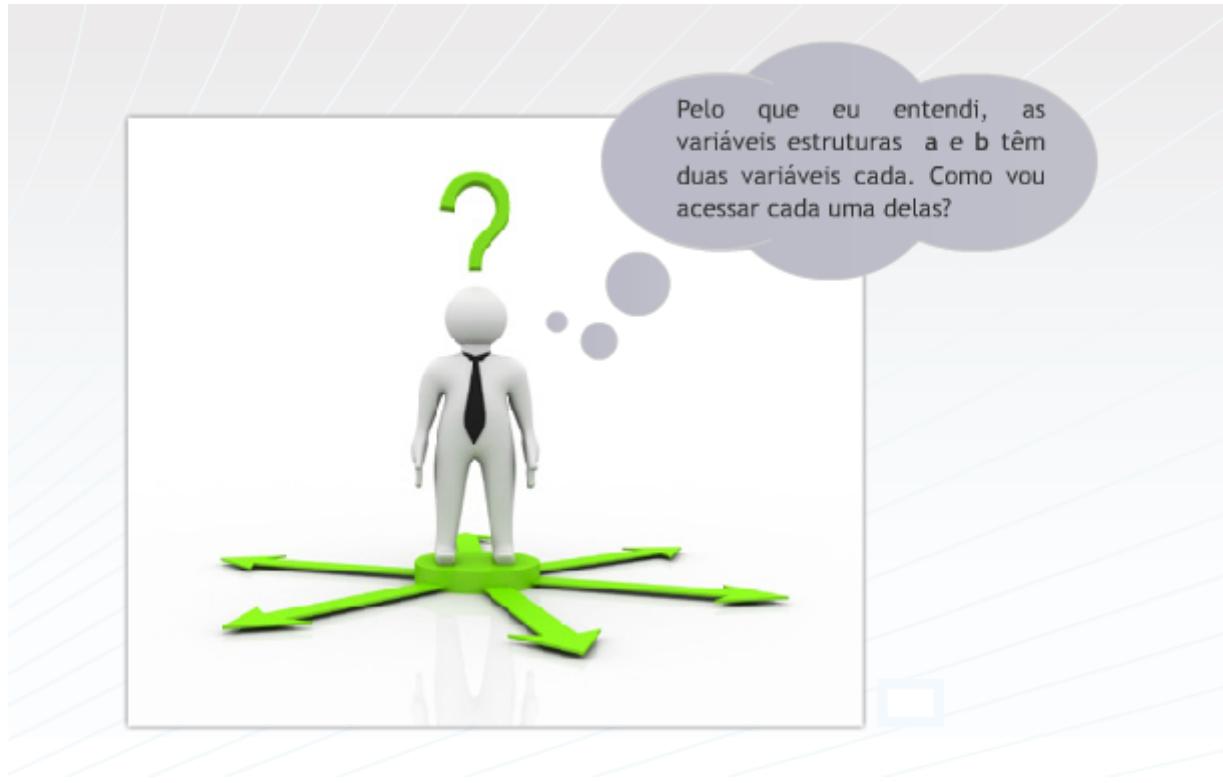
Vamos entender esta estrutura e as duas variáveis declaradas através das figuras abaixo.



A estrutura de nome **coordenadas** tem dois membros do tipo inteiro de nomes **x** e **y**.

As variáveis estruturas **a** e **b** foram declaradas com o tipo **coordenadas** logo, elas terão a mesma composição da estrutura **coordenadas**.





Você está certo. As variáveis de uma estrutura são agrupadas por uma lógica e estão associadas a um mesmo nome. Para acessar cada membro, é muito fácil. Observe o próximo item.

Como se acessa um membro de uma estrutura?

Um membro da estrutura pode ser acessado usando o operador membro das estruturas(.), chamado operador ponto, colocado entre o nome da variável estrutura e o nome do membro.

nomeDaVariávelEstrutura.nomeDoMembro

No nosso exemplo:

Nome da variável estrutura: a

Nome do primeiro membro: x

Nome do segundo membro: y

Acessando o membro x da variável estrutura a: a. x

Acessando o membro y da variável estrutura a: a. y

Nome da variável estrutura: b

Nome do primeiro membro: x

Nome do segundo membro: y

Acessando o membro x da variável estrutura b: b. x

Acessando o membro y da variável estrutura b: b. y

1º. Método

Atribuindo valores aos membros

Usando comando de atribuição na hora da declaração das variáveis.

Atenção para as chaves e para a sequência correta dos dados.

struct coordenadas

{

int x, y;

};

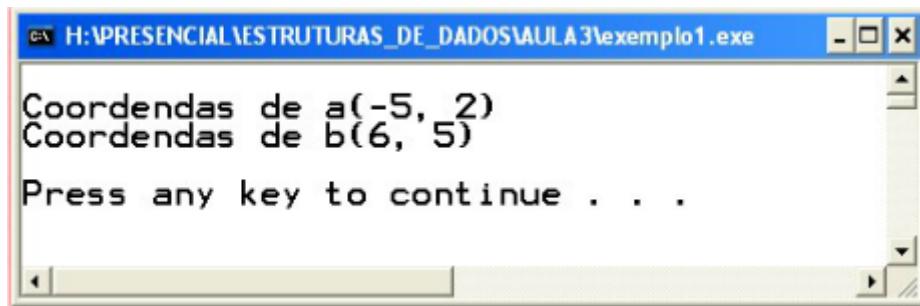
.

coordenadas a={-5, 2};

coordenadas b={ 6, 5};

Observe o código em C++ com a respectiva saída.

```
exemplo1.cpp |  
1 #include <iostream>  
2 using namespace std;  
3 int main()  
4 {  
5     struct coordenadas  
6     {  
7         int x, y;  
8     };  
9     coordenadas a={-5,2}, b={6,5 };  
10    cout<<"\nCoordendas de a("<<a.x<<", "<<a.y<<")";  
11    cout<<"\nCoordendas de b("<<b.x<<", "<<b.y<<")";  
12    cout<<"\n\n";  
13    system("pause");  
14 }
```



2º método

Atribuindo valores aos membros

Atribuição na hora da definição/declaração das variáveis.

Atribuição de uma estrutura a outra estrutura um ponto a favor das estruturas em relação aos vetores.

struct prod

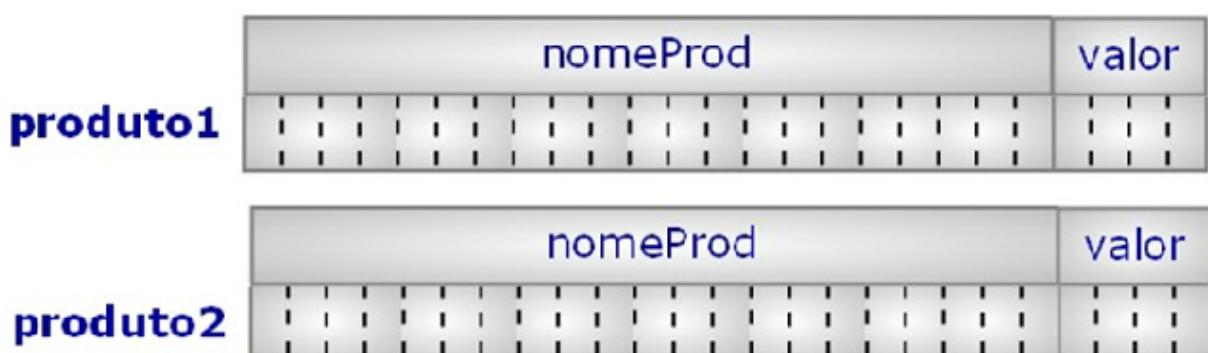
{

```
char nomeProd[21];
```

float valor;

```
produto1={"martelo", 35.90}, produto2={"furadeira", 256.75};
```

Visualizando as variáveis estruturas produto1 e produto2:



Acessando cada membro de cada variável estrutura do tipo *prod*

No nosso exemplo:

Nome da variável estrutura: **produto1**

Nome do primeiro membro: nomeProd

Nome do segundo membro: **valor**

Acessando o membro `nomeProd` da variável estrutura `produto1`: `produto1.nomeProd`

Acessando o membro **valor** da variável estrutura **produto1**: **produto1.valor**

Nome da variável estrutura: **produto2**

Nome do primeiro membro: **nomeProd**

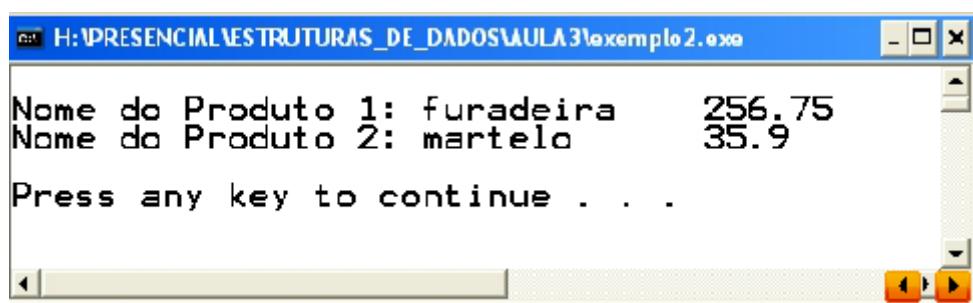
Nome do segundo membro: **valor**

Acessando o membro **nomeProd** da variável estrutura **produto2**: **produto2.nomeProd**

Acessando o membro valor da variável estrutura **produto2**: **produto2.valor**

Observe o código em C++ com a respectiva saída.

```
exemplo2.cpp
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4 int main()
5 {
6     struct prod
7     {
8         char nomeProd[21];
9         float valor;
10    }produto1={"martelo", 35.90}, produto2={"furadeira", 256.75}, aux;
11    if(strcmp(produto1.nomeProd, produto2.nomeProd)>0)
12    { aux=produto1 ; produto1=produto2; produto2 = aux;}
13    cout<<"\nNome do Produto 1: "<<produto1.nomeProd<<"\t"<<produto1.valor;
14    cout<<"\nNome do Produto 2: "<<produto2.nomeProd<<"\t"<<produto2.valor;
15    cout<<"\n\n";
16    system("pause");
17 }
```



Atenção

Neste exemplo, apresentamos uma das grandes vantagens do uso da variável estrutura na programação: a facilidade de se atribuir todo o conteúdo de uma variável estrutura a outra variável estrutura. Você deve estar lembrado que precisávamos usar strcpy para copiar vetores de char e comandos de atribuição para as demais

variáveis. Usando variável estrutura, com um único comando de atribuição, copiamos todos os conteúdos dos membros mesmo que eles sejam vetores de char. Observe como teria ficado o programa sem o uso da variável estrutura. Imagine se a variável estrutura tivesse 10 membros?

```
exemplo2-sem-Estrutura.cpp
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4 int main()
5 {
6     char nomeProd1[21] = "marte lo", nomeProd2[21] = "furadeira", auxC[21] :
7     float valor1=35.90, valor2=256.75, auxf;
8     if(strcmp(nomeProd1,nomeProd2)>0)
9     {
10         strcpy(auxC,nomeProd1); strcpy(nomeProd1,nomeProd2); strcpy(nomeProd2,auxC);
11         auxf=valor1; valor1=valor2; valor2=auxf;
12     }
13     cout<<"\nNome do Produto 1: "<<nomeProd1<<"\t"<<valor1;
14     cout<<"\nNome do Produto 2: "<<nomeProd2<<"\t"<<valor2;
15     cout<<"\n\n";
16     system("pause");
17 }
```

Atribuindo valores aos membros

3º método

Atribuição através da leitura via teclado

```
struct prod
{
    char nomeProd[21];
    float valor;
}produto1, produto2;
```

Observe o código em C++ com a respectiva saída.

```
Nome do primeiro produto: chave de fenda
Valor: 35.70
Nome do segundo produto: chave de boca
Valor: 46.90
Nome do Produto 1: chave de boca      46.9
Nome do Produto 2: chave de fenda      35.7
Press any key to continue . . .
```

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main()
5 {
6     struct Prod
7     {
8         char nomeProd[31];
9         float valor;
10    } prod1, prod2;
11    cout << "Nome do primeiro produto: ";
12    cin >> prod1.nomeProd >> prod1.valor;
13    cout << "Valor: ";
14    cin >> prod1.valor;
15    cout << endl; //retira o enter
16    cout << "Nome do segundo produto: ";
17    cin >> prod2.nomeProd >> prod2.valor;
18    cout << endl;
19    cout << prod1.nomeProd << prod1.valor;
20    cout << endl;
21    if(prod1.valor > prod2.valor)
22        cout << "O menor valor é: " << prod2.valor;
23    else
24        cout << "O menor valor é: " << prod1.valor;
25    cout << endl;
26    system("pause");
27 }
```

Baixe a versão TXT dos códigos **AQUI**:

http://estaciодocente.webaula.com.br/cursos/gon119/conteudo/docs/aula03_11.txt

Agora é a sua vez!

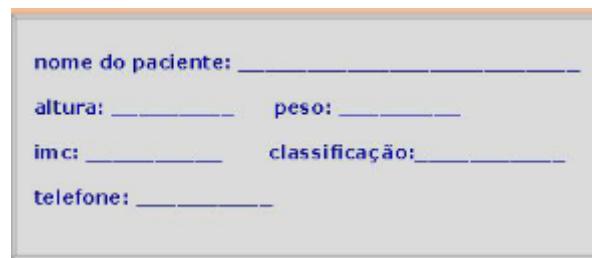
Você deverá construir programas, usando estruturas. Em cada um dos exemplos, é solicitada uma forma diferente para atribuir valores aos membros das estruturas. Lembre-se de que você é o programador e a saída não precisa ser igual a da ficha. Use sua criatividade, mas primeiro faça sua solução e, só depois, pressione o botão Solução para comparar as soluções. Não use vetores ainda.

Exemplo 1

Observe a ficha abaixo e defina uma estrutura que armazene todos os dados dela.

Declare variáveis, supondo dois pacientes e construa um programa, lendo os dados via teclado.

Exiba os dados exceto o telefone



Solução

```
exemplo-AJUNO.cpp
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     //definição da estrutura
6     struct cad
7     {
8         char nomePaciente[31],classificacao[25], telefone[14];
9         float altura, peso, imc;
10    };
11    cad paciente1, paciente2; //declaração das variáveis estruturas
12    // primeiro paciente
13    cout<<"\nNome do 1º paciente: "; cin.getline(paciente1.nomePaciente, 31);
14    cout<<"\nDigite Peso, Altura e IMC, pressionando enter após cada um\n";
15    cin>>paciente1.peso>>paciente1.altura>>paciente1.imc;
16    cout<<"\nDigite a classificação: "; cin.getline(paciente1.classificacao, 25);
17    cout<<"\nDigite telefone: "; cin.getline(paciente1.telefone, 25);
18
19    // segundo paciente
20    cout<<"\nNome do 2º paciente: "; cin.getline(paciente2.nomePaciente, 31);
21    cout<<"\nDigite Peso, Altura e IMC, pressionando enter após cada um\n";
22    cin>>paciente2.peso>>paciente2.altura>>paciente2.imc;
23    cin.get(); //retira o enter deixado pela variável numerica
24    cout<<"\nDigite a classificação: "; cin.getline(paciente2.classificacao, 25);
25    cout<<"\nDigite telefone: "; cin.getline(paciente2.telefone, 25);
26
27    //limpando a tela
28    system("cls"); //system("clear"); no LINUX
29    cout<<"\nNome do 1º Paciente: "<<paciente1.nomePaciente;
30    cout<<"\nPeso\Altura\tIMC\tClassificação";
31    cout<<"\n"<<paciente1.peso<<"\t"<<paciente1.altura<<"\t"<<paciente1.imc<<"\t"<<paciente1.classificacao<<"\n";
32    cout<<"\nNome do 2º Paciente: "<<paciente2.nomePaciente;
33    cout<<"\nPeso\Altura\tIMC\tClassificação";
34    cout<<"\n"<<paciente2.peso<<"\t"<<paciente2.altura<<"\t"<<paciente2.imc<<"\t"<<paciente2.classificacao<<"\n";
35    system("pause");
36 }
```

Clique para expandir

```
C:\Documents and Settings\Anita Lopes\Desktop\EDAD-ED\aula3... - □ x

None do 1o Paciente: Joao
Peso   Altura   IMC      Classificao
65     1.8       20.6    normal

None do 2o Paciente: Pedro
Peso   Altura   IMC      Classificao
100    1.7       34.6    Obesidade III
Press any key to continue . . .
```

Exemplo 2

Observe a ficha abaixo e defina uma estrutura que armazene todos os dados dela.

Declare variáveis, supondo três passageiros e construa um programa, atribuindo os valores dos membros na definição/declaração.

nome do passageiro: _____
origem: _____ destino: _____
número da passagem: _____
identidade: _____ telefone: _____

Solução

```
exemplo2\ALUNO.cpp |  
1 #include <iostream>  
2 using namespace std;  
3 int main()  
4 { //definicao da estrutura  
5 struct aviao  
6 {  
7     char nomePassageiro[21], origem[15], destino[15], numeroPassagem[15], identidade[15], telefone[15];  
8 }passageiro1={"Mr Lopes, Joao", "Brasil", "Londres", "Gol 1234", "IIP222222222", "2123333333"},  
9 passageiro2={"Mrs Lopes, Tereza", "Brasil", "Paris", "Gol 1234", "IIP777777777", "2123344444"},  
10 passageiro3={"Mr Ferreira, Marcelo", "Brasil", "Lisboa", "Gol 1239", "IIP23704567", "2234567890"},  
11  
12 //limpando a tela  
13 system("cls"); //system("clear"); no LINUX  
14 cout<<"\n+++++++\n";  
15 cout<<"\nNome: "<<passageiro1.nomePassageiro;  
16 cout<<"\nOrigem: "<<passageiro1.origem<<"\t\tDestino: "<<passageiro1.destino;  
17 cout<<"\nNúmero da passagem: "<<passageiro1.numeroPassagem<<"\tIdentidade: "<<passageiro1.identidade<<"\tTelefone: "<<passageiro1.telefone;  
18 cout<<"\n+++++++\n";  
19  
20 cout<<"\n+++++++\n";  
21 cout<<"\nNome: "<<passageiro2.nomePassageiro;  
22 cout<<"\nOrigem: "<<passageiro2.origem<<"\t\tDestino: "<<passageiro2.destino;  
23 cout<<"\nNúmero da passagem: "<<passageiro2.numeroPassagem<<"\tIdentidade: "<<passageiro2.identidade<<"\tTelefone: "<<passageiro2.telefone;  
24 cout<<"\n+++++++\n";  
25  
26 cout<<"\n+++++++\n";  
27 cout<<"\nNome: "<<passageiro3.nomePassageiro;  
28 cout<<"\nOrigem: "<<passageiro3.origem<<"\t\tDestino: "<<passageiro3.destino;  
29 cout<<"\nNúmero da passagem: "<<passageiro3.numeroPassagem<<"\tIdentidade: "<<passageiro3.identidade<<"\tTelefone: "<<passageiro3.telefone;  
30 cout<<"\n+++++++\n";  
31 cout<<"\n";  
32 system("pause");  
33 }
```

```
C:\Documents and Settings\Anita Lopes\Desktop\EAD-ED\aula3_anita\exemplo2-aluno.exe
*****
Nome: Mr Lopes, Joao          Destino: Londres
Origen: Brasil     Numero da passagem: Gol 1234   Identidade: IFP22222222 Telefone: 2123333333
*****
*****
Nome: Mrs Lopes, Tereza        Destino: Paris
Origen: Brasil    Numero da passagem: Gol 1234   Identidade: IFP77777777 Telefone: 2123344444
*****
*****
Nome: Mr Ferreira, Marcelo    Destino: Lisboa
Origen: Brasil   Numero da passagem: Gol 1239   Identidade: IFP23784567 Telefone: 2234567890
*****
Press any key to continue . . .
```

Exemplo 3

Observe a ficha abaixo e defina uma estrutura que armazene todos os dados dela.

Declare variáveis, supondo 1 atleta e construa um programa, atribuindo os valores dos membros, um a um, após a declaração.

nome do atleta:	<input type="text"/>		
esporte:	<input type="text"/>	categoria:	<input type="text"/>
valor da ajuda de custo:	<input type="text"/>		
telefone:	<input type="text"/>	ano_Nas:	<input type="text"/>

Solução

```

exemplo3ALUNO.cpp | 
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4 int main()
5 { //definição da estrutura
6     struct cadastro
7     {
8         char nomeAtleta[31],esporte[25], categoria[15],telefone[15];
9         float ajudaCusto;
10        int anoNas;
11    };
12    cadastro atleta; // declarar atleta como uma variável tipo cadastro
13    //atribui valores dos membros
14    strcpy(atleta.nomeAtleta, "Joao Lopes");
15    strcpy(atleta.esporte, "Volley");
16    strcpy(atleta.categoria, "Adulto");
17    strcpy(atleta.telefone, "21-22334455");
18    atleta.ajudaCusto=2000;
19    atleta.anoNas=1989;
20
21    //limpando a tela
22    system("cls"); //system("clear"); no LINUX
23    cout<<"\n=====";
24    cout<<"\n= Nome : "<<atleta.nomeAtleta<<"      =";
25    cout<<"\n= Esporte: "<<atleta.esporte<<"\t\tCategoria: "<<atleta.categoria<<"      =";
26    cout<<"\n= Valor da ajuda de custo : "<<atleta.ajudaCusto<<"      =";
27    cout<<"\n= Telefone: "<<atleta.telefone<<"\t\tAno_nas: "<<atleta.anoNas<<"      =";
28    cout<<"\n=====";
29
30    cout<<"\n";
31    system("pause");
32 }

```



Não se esqueça: não existe uma solução única para um problema.

2 Construindo um array de estruturas

Na disciplina de Algoritmos, nós estudamos que um array é um conjunto de variáveis todas do mesmo tipo.

Talvez possa causar certa estranheza estarmos falando em array de estruturas, mas pense de forma mais ampla: as estruturas não são iguais? Sendo assim, construir um conjunto de estruturas está correto.

Como se declara um array de estruturas?

1º método

Da mesma forma que declaramos as estruturas simples, podemos declarar os arrays de estruturas. A declaração é feita depois da definição. Se a definição for global, a declaração acontece em cada função.

2º método

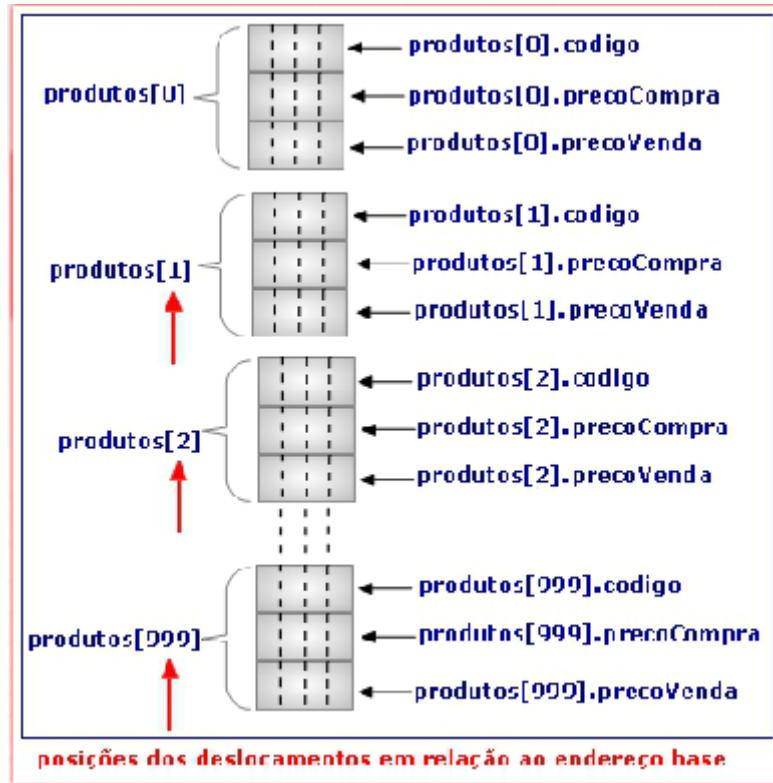
A declaração é feita junta com a definição.

<pre>struct <identificador> { ... }; <identificador> nomedaVariável;</pre>	<pre>struct a/Cad { ... }; a/Cad alunos[15];</pre>
<pre>struct <identificador> { ... } nomedaVariável;</pre>	<pre>struct a/Cad { ... } a/Cad alunos[15];</pre>

3 Visualizando um array de estruturas e acessando um elemento do array

Suponha a seguinte declaração de um array de estruturas e sua visualização.

```
struct prodCad
{
    int codigo;
    float precoCompra, precoVenda;
};
prodCad produtos[1000];
```



4 Acessando um elemento do array

Precisamos ter muito cuidado quando formos acessar um array de estruturas porque primeiro temos que escolher a variável do array como fazímos com os arrays unidimensionais. A figura esclarece isso do lado esquerdo quando relaciona as variáveis estruturas com um par de colchetes e o deslocamento dentro dele (índice).

Depois, acrescentamos separado por um ponto, os membros da estrutura. Você poderá ver isso no lado direito da figura.

Cada retângulo está dividido por linhas tracejadas porque uma variável do tipo int, ou do tipo float, ocupa quatro posições de memória.

Atenção

Gostaríamos de relembrar que os dados são alocados de forma contígua na Memória Principal como se fosse uma matriz unidimensional. A figura acima é meramente ilustrativa e tem como objetivo possibilitar uma melhor compreensão sobre um array de estruturas.

5 Algumas considerações

Num primeiro momento, é normal se questionar: por que não continuamos usando arrays unidimensionais. Não seria mais fácil? Será?

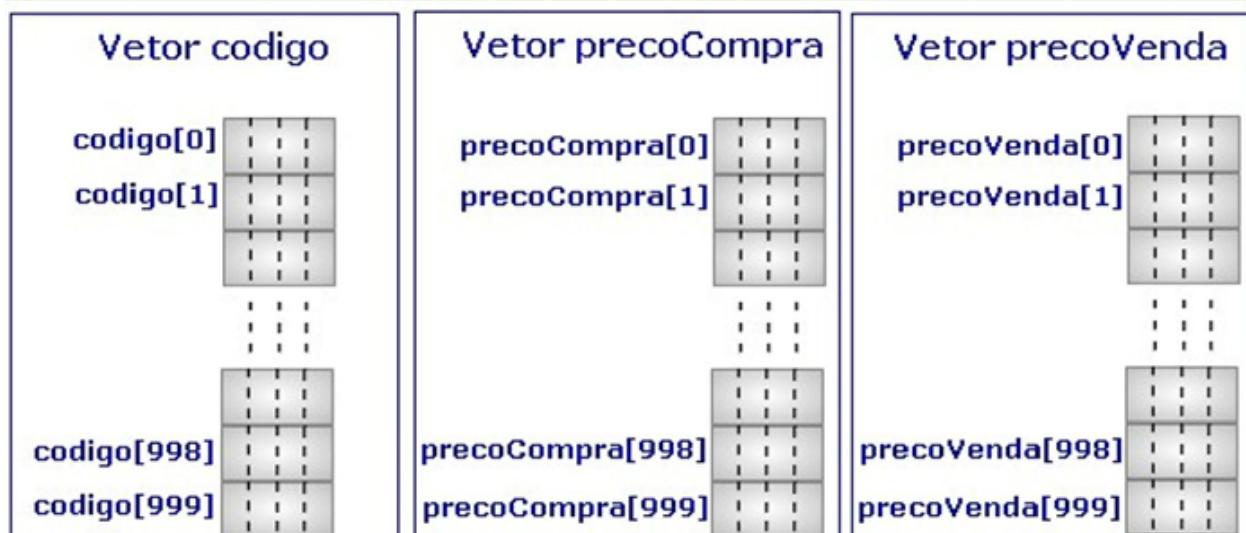
Você se lembra, logo no início, quando perguntava ao professor ou a um colega: “como o computador sabe que a idade é dessa pessoa e não da outra?” e a pessoa lhe respondia: “Não sabe, pois os nomes das variáveis têm que ter sentido para você.” E, muitas vezes, você ficava sem entender, mas, com o passar do tempo, foi percebendo que se tivesse que criar três variáveis para guardar nomes de pessoas e três variáveis para guardar as idades delas, você escolhia algo parecido com: nome1, nome2, nome3, idade1, idade2, idade3.

Até que um dia, você aprendeu matrizes e começou a se questionar novamente como as matrizes poderiam se relacionar. Enfim, este dia chegou. Usando um array de estruturas.

Antes desta aula, dimensionaríamos três arrays unidimensionais (vetores) ou um array unidimensional inteiro e um array bidimensional real.

```
int codigo[1000];float precoCompra[1000], precoVenda[1000]; ou  
int codigo[1000];float precos[1000][2];
```

Veja como ficaria se tivéssemos feito a opção por três arrays unidimensionais.



Atenção

Temos portanto, duas formas de resolver o problema: uma matriz de estrutura ou três matrizes. Se preferirmos a segunda, percebe-se que não existe uma relação entre as matrizes e quando formos ordenar, precisaremos de muitos trechos para trocar as variáveis de lugar. Não que esteja errado, mas tudo ficaria mais extenso se continuássemos usando somente as matrizes. Lembre-se que elas não desaparecem, mas se agregam.

Agora veja três exemplos diferentes:

Exemplo 1

Tomando como base o array de estruturas, construa um programa que armazene código e preço de compra de 1000 produtos, calcule um reajuste de 60% para venda e exiba código, valor de compra e venda. Observação: Para que você pudesse visualizar a entrada e a saída, foi usada a diretiva define que criou a constante TAM com valor 3 (deveria ser 1000).

```
Exemplo0.Array.cpp  x
1 #include <iostream>
2 #define TAM 3
3 using namespace std;
4 int main()
5 {
6     struct prodCad
7     {
8         int codigo;
9         float precoCompra, precoVenda;
10    };
11    prodCad produtos[TAM];
12    int x, num;
13    for(x=0; x<TAM;x++)
14    {
15        cout<<"\nCodigo do Produto 0-10000: ";
16        cin>num;
17        while(num<0 || num>10000)
18        {
19            cout<<"\nCodigo Invalido";
20            cout<<"\nCodigo do Produto 0-10000: ";
21            cin>num;
22        }
23        produtos[x].codigo=num;
24        cout<<"\nPreco de compra: ";
25        cin>>produtos[x].precoCompra;
26        produtos[x].precoVenda= produtos[x].precoCompra *1.6;
27    }
28    system("cls");
29    cout<<"\n\nCodigo\tCompra\tVenda\n";
30    for(x=0; x<TAM;x++)
31    cout<<"\n"<<produtos[x].codigo<<"\t"<<produtos[x].precoCompra<<"\t"<<produtos[x].precoVenda;
32    cout<<"\n\n";
33    system("pause");
34 }
```



Codigo	Compra	Venda
238	200	320
1289	300	480
1989	500	800

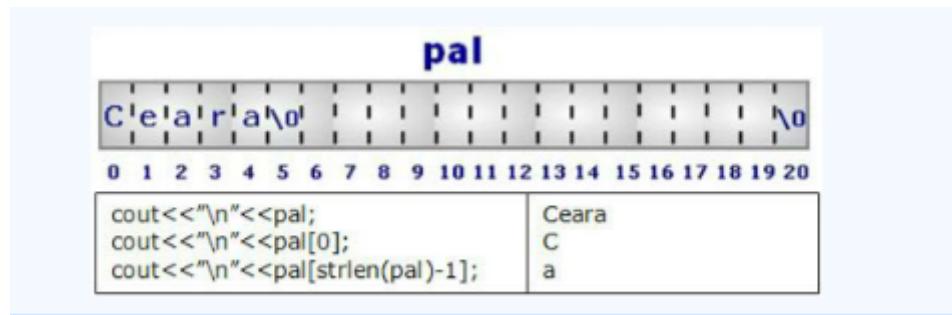
Exemplo 2

Construa um array de estruturas com duas variáveis. O único membro da estrutura é um array de caracter (vetor de char) que deverá ser capaz de armazenar até 20 caracteres sem incluir o terminador. Os valores deverão ser lidos via teclado. Na saída, cada letra de cada vetor de char deverá ser exibida em uma linha, sendo que uma linha em branco separará as letras dos dois vetores. Além disso, a primeira letra da primeira palavra tem que estar em maiúscula mesmo que o usuário não tenha digitado assim.

Algumas considerações sobre esse exercício:

Parece que o mais complicado para alguns alunos é a manipulação do vetor de char e quando ele é um membro da estrutura, parece que se torna ainda mais difícil.

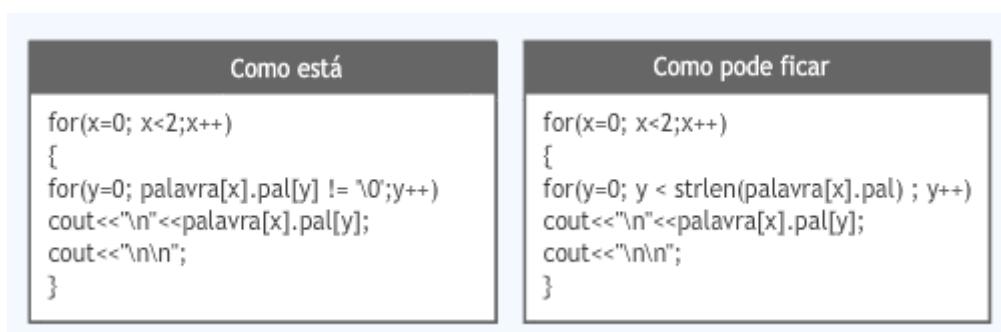
Na maioria dos exemplos anteriores, já tivemos como membro de uma estrutura, um vetor de char. Além disso, estudamos na disciplina de Algoritmos que o vetor de char tem um tratamento diferenciado dos demais vetores unidimensionais, pois ele é declarado como um vetor e tratado como uma variável simples exceto quando precisamos manipular uma letra de cada vez. Observe a visualização de um vetor de char e as saídas que obteremos com os comandos abaixo. Atente para os parênteses usados quando desejamos exibir um caracter do vetor.



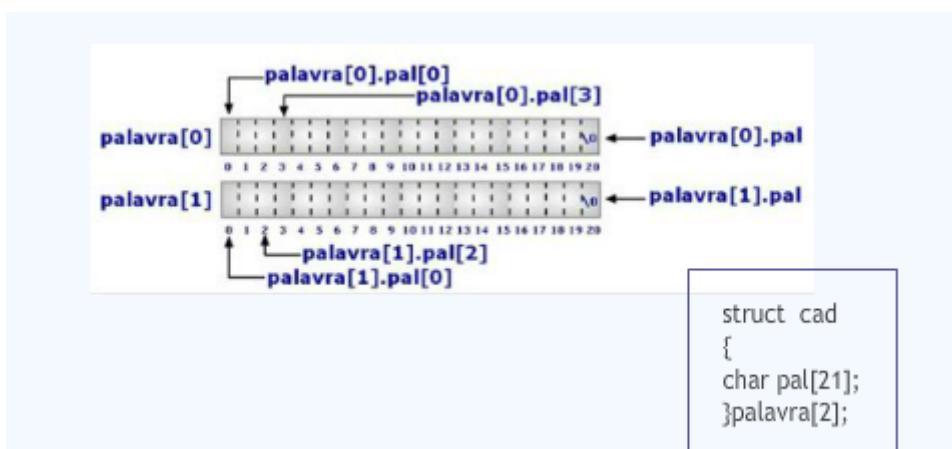
Você terá que usar a função **toupper()** da biblioteca **cctype** para converter para maiúscula a primeira letra da primeira palavra. Observe a linha 36 do código fonte.

Você poderá refazer o trecho que exibe uma letra em cada linha, usando a função **strlen()** da biblioteca *cstring*. Preferimos controlar **usando** o terminador \0.

Observe que a estrutura do for não está sendo controlada pela variável y logo, neste exemplo, está sendo usada como se fosse a estrutura *while*. Experimente trocar pelo trecho abaixo e veja se faz alguma diferença.



Nosso array só terá dois elementos para que você possa visualizar a saída. Observe a definição/ declaração e a visualização do array de estruturas.



Exemplo 3

Reforçando os conceitos

A estrutura criada tem nome : cad.

A estrutura só tem um membro cujo nome é: pal.

O nome do array de estruturas do tipo cad é: palavra.

Esse array tem duas variáveis estruturas: palavra[0] e palavra[1].

Para acessar o único membro da estrutura, a variável palavra[0] precisa ser unida por um . ao membro pal, ficando assim: palavra[0].pal.

Da mesma forma, palavra[1].pal.

Quando usarmos um dos nomes acima com cout, exibirá na tela todo o conteúdo de um dos elementos do vetor.

Quando desejarmos acessar uma letra de uma das variáveis estruturas, precisaremos usar a seguinte sintaxe:

nomeDoArray[deslocamento array].membro[deslocamento na variável]

Exemplo:

Se você desejar a terceira letra da segunda palavra: palavra[1].pal[2].

The screenshot shows a code editor window titled "Exemplo1.Array.cpp". The code is as follows:

```
1 #include <iostream>
2 #include <cctype>
3 using namespace std;
4 int main()
5 {
6     struct cad
7     {
8         char pal[21];
9     }palavra[2];
10    int x,y;
11    for(x=0; x<2;x++)
12    {
13        cout<<"\nPalavra: ";
14        cin.getline(palavra[x].pal, 21);
15    }
16    palavra[0].pal[0]=toupper(palavra[0].pal[0]);
17    system("cls");
18    for(x=0; x<2;x++)
19    {
20        for(y=0;palavra[x].pal[y] != '\0';y++)
21            cout<<"\n"<<palavra[x].pal[y];
22        cout<<"\n\n";
23    }
24    cout<<"\n\n";
25 }
26 }
```

The code defines a structure "cad" containing a character array "pal" of size 21. It then creates two instances of this structure in an array "palavra". The main function reads two lines of input from the user, each containing a word. It then converts the first character of the first word to uppercase and prints all characters of both words, each on a new line.

```

I:\MULAS_ANITA\exemplo1Ar...
Palavra: ceara
Palavra: verao

I:\MULAS_ANITA\exemplo1Ar...
c
e
a
r
a
v
e
r
a
o

Press any key to continue . . .

```

Faça o download dos arquivos aqui:

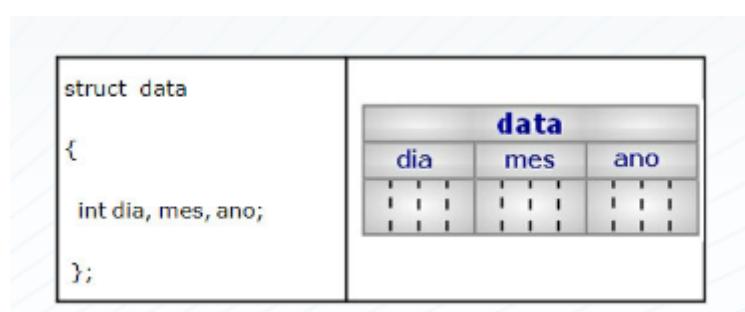
http://estaciocente.webaula.com.br/cursos/gon119/docs/a03_t16.rar

6 Estruturas aninhadas

Muitas vezes, definimos estruturas que podem ser membros de várias estruturas.

Em qualquer cadastro, por exemplo, tem data: data de nascimento, data de pagamento, data de recebimento, data de entrega, etc.

Sendo assim, poderíamos definir uma estrutura data e visualizá-la da seguinte forma:

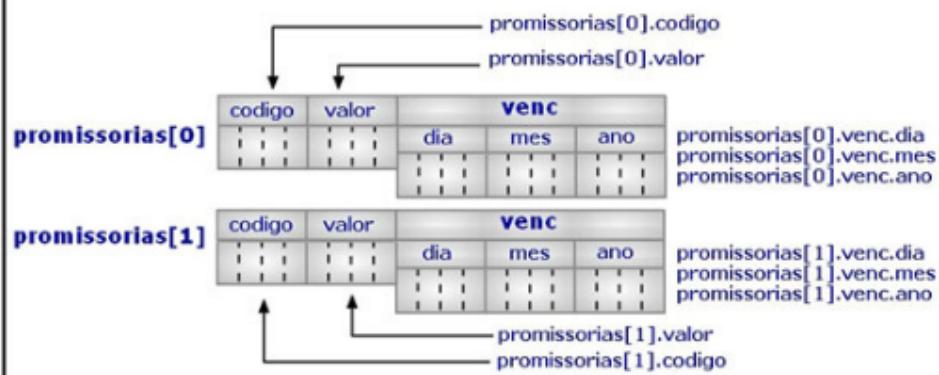


E usarmos esta estrutura com membro de outra estrutura que iremos definir/declarar e visualizar assim:

```

struct pago
{
    int codigo;
    float valor;
    data venc;
}promissorias[2];

```



Acessando um membro de uma estrutura dentro de um array de estruturas

nomeDoArray[deslocamento no array].nomeDaEstruturaAninhada.membro

Exemplo

Acessando o dia da variável estrutura de deslocamento 1 = `promissorias[1].venc.dia`

Vamos construir um programa para armazenar valores nesse array de estruturas aninhadas que declaramos acima.

```

entituturasAnhadas.cpp
1 #include <iostream>
2 using namespace std;
3 struct data
4 {
5     int dia, mes, ano;
6 };
7 int main()
8 {
9     struct pago
10    {
11        int codigo;
12        float valor;
13        data venc;
14    }promissorias[2];
15    int x;
16    for(x=0; x<2;x++)
17    {
18        cout<<"\nCodigo: ";
19        cin>>promissorias[x].codigo;
20        cout<<"\nValor a ser pago: ";
21        cin>>promissorias[x].valor;
22        cout<<"\nDigite dia :";
23        cin>>promissorias[x].venc.dia;
24        cout<<"\nDigite mes :";
25        cin>>promissorias[x].venc.mes;
26        cout<<"\nDigite ano :";
27        cin>>promissorias[x].venc.ano;
28    }
29    system("cls");
30    cout<<"\nCodigo\tValor\tData de Vencimento\n";
31    for(x=0; x<2;x++)
32    cout<<"\n" << promissorias[x].codigo << "\t" << promissorias[x].valor << "\t" << promissorias[x].venc.dia << "/" << promissorias[x].venc.mes << "/" << promissorias[x].venc.ano;
33    cout<<"\n\n";
34    system("pause");
35 }
36
37

```

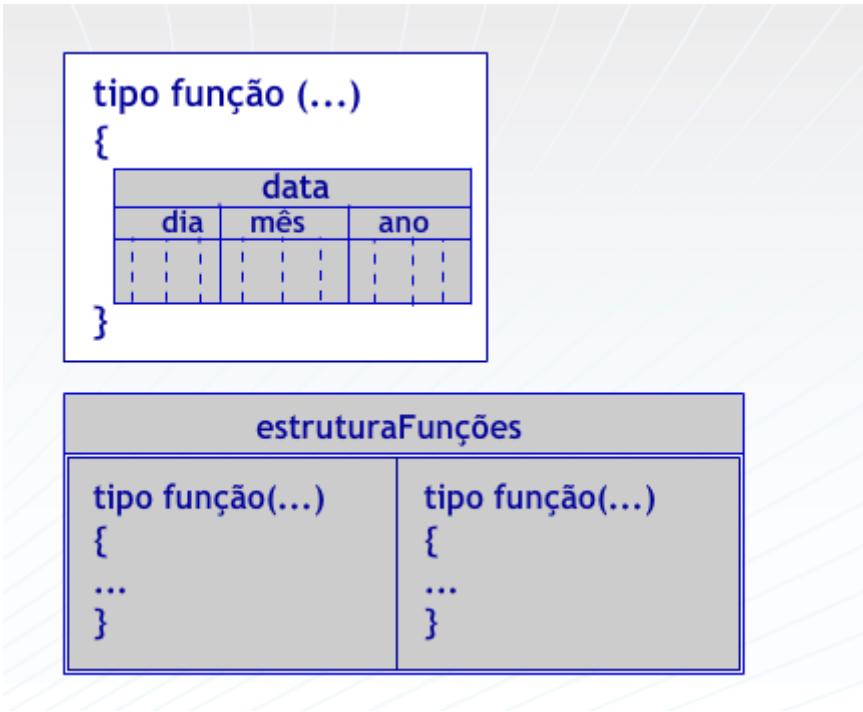
Atenção

Observe que são dois pontos. O primeiro entre promissorias[1] e o membro venc. Mas como venc é uma estrutura e dia é membro de venc tem um ponto separando venc de dia. Esse é o segundo ponto.

Codigo	Valor	Data de Vencimento
13	3000	13/ 8 /2011
23	1800	23/ 3 /2011

Press any key to continue . . .

O encontro das estruturas com as funções e das funções com as estruturas.



Atenção

Na aula passada, estudamos que o uso de funções melhora a legibilidade e a manutenibilidade do programa logo, chegou o momento de criarmos funções com/para estruturas.

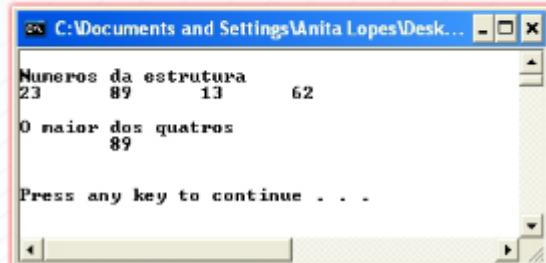
Você verá algumas coisas interessantes que poderemos fazer ao construirmos funções com estruturas como por exemplo: estruturas poderão ser retornos de função, estruturas poderão conter função dentro delas além de tudo que já foi feito com os arrays homogêneos.

7 Passagem por valor

Da mesma forma que passamos o conteúdo de uma variável simples ou de um elemento do vetor, podemos passar um membro (campo) de uma estrutura. Vale a pena relembrar que, na passagem por valor, passamos uma cópia do conteúdo logo, o valor original não se altera.

Neste exemplo, chamaremos a função três vezes, passando os membros da estrutura de tal maneira que retorne o maior dos quatro números.

A função
<pre>float maior2(float n1, float n2) { if(n1>n2) return n1; else return n2; }</pre>
A chamada da função
count << "\n\t" << maior2(maior2(numero.a.numero.b),maior2(numero.c.numero.d));



```
funcaoValor.cpp |
1 #include <iostream>
2 using namespace std;
3
4 struct cad           // definição de cad
5 {
6     float a,b,c,d;
7
8 };
9
10 float maior2(float n1, float n2);
11
12 int main()
13 {
14     cad numeros={23, 89,13, 62};           // declaração/atribuição de numeros
15     cout<<"\nNumeros da estrutura";
16     cout << "\n" << numeros.a << "\t" << numeros.b << "\t" << numeros.c << "\t" << numeros.d;
17     cout << "\n\nO maior dos quatros";
18     cout << "\n\t" << maior2(maior2(numero.a,numero.b),maior2(numero.c,numero.d));
19     cout << "\n\n\n";
20     system("pause");
21     return(0);
22 }
23
24 float maior2(float n1, float n2)
25 {
26     if(n1> n2) return n1;
27     else return n2;
28 }
```

8 Passagem por referência

Quando estudamos funções, para quem já tinha visto a linguagem C, notou que a passagem por referência simplificou o uso da passagem por endereço(ponteiros). Por isso, citamos SAADE, Joel(2003, p.112): "Pode-se dizer que a passagem por referência é uma forma disfarçada de ponteiro".

Neste exemplo, passaremos o endereço de um membro do struct por referência. A função irá calcular a idade da pessoa em 2020 e irá alterar o valor do membro.

```
funcaoReferencia.cpp |  
1 #include <iostream>  
2 using namespace std;  
3  
4 struct cad           // definição de cad  
5 {  
6     char nome[31];  
7     int idade;  
8 };  
9  
10 void idade2020(int &id);  
11  
12 int main()  
13 {  
14     cad atleta={"Joao Renato", 21};           // declaração/atribuição de Atleta  
15     cout<<"\nAntes da Chamada da Funcao";  
16     cout << "\nNome do atleta: "<< atleta.nome << "\tIdade em 2010: " << atleta.idade;  
17     idade2020(atleta.idade);  
18     cout<<"\n\nDepois da Chamada da Funcao";  
19     cout << "\nNome do atleta: "<< atleta.nome << "\tIdade em 2020: " << atleta.idade;  
20     cout << "\n\n";  
21     system("pause");  
22     return(0);  
23 }  
24  
25 void idade2020(int &id)  
26 {  
27     id=id+10;  
28 }
```

A função
void idade2020(int&id) { id=id+10; }
A chamada da função
idade2020(atleta.idade);



9 Passagem de um membro que é um vetor - passagem por endereço

Nós já vimos que os arrays são, em verdade, endereços tanto é que o que fica entre parênteses é o deslocamento em relação ao endereço base. Sendo assim, a passagem de um membro que é um vetor se dá de mesma forma que fizemos com os arrays unidimensionais. Só para reforçar esse conceito que consideramos muito importante, vamos observar o programa abaixo. Nele, fizemos referência ao nome da estrutura e aos nomes das três variáveis do array, mas somente os nomes das três variáveis foram precedidos pelo símbolo & que representa endereço.

```
funcaoMembroEnderecos.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     struct cad           // definição de cad
7     {
8         char nome[31];
9     };
10    cad funcionario[3]; // declaração de funcionario
11    //saída
12    cout<<"\n\nEndereco da estrutura( nem coloquei o &): "<<funcionario;
13    cout<<"\n\nEndereco da 1a variável da estrutura      : "<<&funcionario[0];
14    cout<<"\n\nEndereco da 2a variável da estrutura      : "<<&funcionario[1];
15    cout<<"\n\nEndereco da 3a variável da estrutura      : "<<&funcionario[2];
16    cout << "\n\n";
17    system("pause");
18    return(0);
19 }
```

```
C:\Documents and Settings\Anita Lopes\Desktop\EAD-EDA...
Endereco da estrutura( nem coloquei o &): 0x22ff10
Endereco da 1a variável da estrutura      : 0x22ff10
Endereco da 2a variável da estrutura      : 0x22ff2f
Endereco da 3a variável da estrutura      : 0x22ff4e
Press any key to continue . . .
```

Na saída, percebemos que o endereço da estrutura é o mesmo da primeira variável do array de estruturas algo que já dissemos desde a disciplina de Algoritmos quando explicamos porque o valor do primeiro elemento do array era 0(zero).

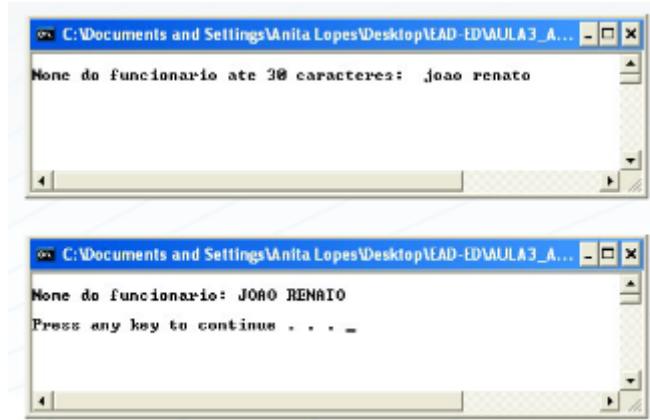
O endereço da segunda variável está distante de 31 posições, visto que, como está representado em hexadecimal, o final 10 seria equivalente em decimal a 16(1 x 16 + 0) enquanto 2f seria equivalente a 47(2 x 16 +15). Então, 47 -16=31 que é o tamanho declarado para o vetor de char. Assim como 4e seria equivalente a 78(4 x 16 + 14) então, 78 - 47= 31. c.q.d (como queríamos demonstrar).

Já que provamos que o vetor é um endereço, vamos ao exemplo onde passaremos um membro da estrutura que é um vetor.

A função
<pre>void maiuscula(char func[]) { int x; for(x=0; func[x] !='0'; x++) func[x]=toupper(func[x]); }</pre>
A chamada da função
maiuscula(funcionario.nome);

```

1 #include <iostream>
2 #include <ctype.h>
3 using namespace std;
4
5 struct cad           // definindo estrutura
6 {
7     char nome[30];      int x;
8 };
9         for(x=0; func[x] !='0'; x++)
10 void maiuscula(char func[]): func[x]=toupper(func[x]);
11 }
12 int main()
13 {
14     cad funcionario;      // declaração de funcionário
15     char lixo[100];
16     cout << "\nNome do funcionário até 30 caracteres: ";
17     cin.getline(lixo,100);
18     while(strlen(lixo)>30)
19     {
20         cout<<"\nNome muito extenso!\n";
21         cout << "\nNome do funcionário até 30 caracteres: ";
22         cin.getline(lixo,100);
23     }
24     strcpy(funcionario.nome, lixo);
25     maiuscula(funcionario.nome);
26     // saída
27     system("cls");
28     cout << "\nNome do funcionário: "<< funcionario.nome ;
29     cout << "\n\n";
30     system("pause");
31     return(0);
32 }
33 void maiuscula(char func[])
34 {
35     int x;
36     for(x=0; func[x] !='0'; x++)
37         func[x]=toupper(func[x]);
38 }
```



```
C:\Documents and Settings\Anita Lopes\Desktop\EAD-ED\aula3_A... -> Nome do Funcionario ate 30 caracteres: joao renato
C:\Documents and Settings\Anita Lopes\Desktop\EAD-ED\aula3_A... -> Nome do funcionario: JOAO RENATO
Press any key to continue . . .
```

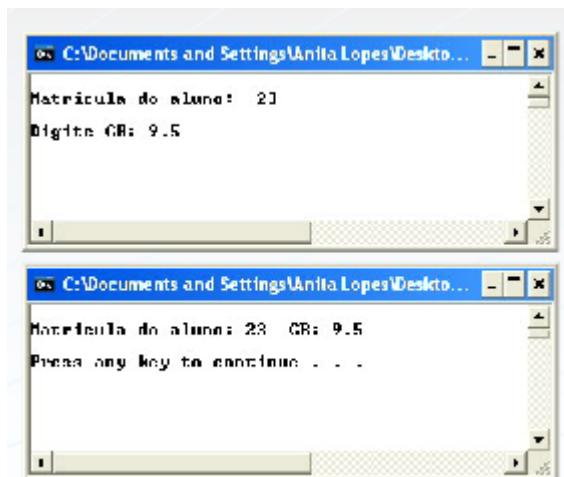
10 Estrutura como parâmetro de uma função

Neste exemplo, apresentaremos uma função que recebe uma estrutura para ser manipulada dentro da função.

A função
<pre>void Exibe(cad AL) { system("cls"); cout << "\nMatricula do aluno: " << AL.matricula << "\tCR: " << AL.CR; cout << "\n\n"; system("pause"); }</pre>

A chamada da função
<pre>Exibe(aluno);</pre>

```
luncaoRecebeEstrutura.cpp |  
1 #include <iostream>  
2 using namespace std;  
3  
4 struct cad           // definição de cad  
5 {  
6     int matricula;  
7     float CR;  
8 };  
9  
10 void Exibe(cad AL); // função com parâmetro que é uma estrutura  
11  
12 int main()  
13 {  
14     cad aluno;      // declaração de aluno  
15     cout << "\nMatrícula do aluno: ";  
16     cin >> aluno.matricula;  
17     cout << "\nDigite CR: ";  
18     cin >> aluno.CR;  
19     Exibe(aluno); // chama a função  
20     // saída  
21     return(0);  
22 }  
23 void Exibe(cad AL)  
24 {  
25     system("cls");  
26     cout << "\nMatrícula do aluno: " << AL.matricula << "\tCR: " << AL.CR;  
27     cout << "\n\n";  
28     system("pause");  
29 }
```



11 Estrutura como valor de retorno de função

Neste exemplo, apresentaremos uma função do tipo estrutura logo, terá como retorno, uma estrutura.

<pre>A função</pre> <pre>cad Entrada() { cad temp; cout << "\nMatrícula do aluno: "; cin >> temp.matricula; cout << "\nDigite CR: "; cin >> temp.CR; return(temp); // retorna uma estrutura }</pre>
<pre>A chamada da função</pre> <pre>aluno = Entrada();</pre> <p>a estrutura aluno recebe o retorno da função Entrada</p>

```
C:\Documents and Settings\Aluna Lopes\Desktop... Matrícula do aluno: 13 Digite CR: 10
```

```
C:\Documents and Settings\Aluna Lopes\Desktop... Matrícula do aluno: 13 CR: 10 Press any key to continue . . .
```

```
funcãoRetornoEstrutura.cpp |
```

```
1 #include <iostream>
2 using namespace std;
3
4 struct cad           // definição de cad
5 {
6     int matricula;
7     float CR;
8 };
9
10 cad Entrada();
11 // protótipo da função Entrada sem参metros,
12 // do tipo cad que é um struct e com retorno
13
14 int main()
15 {
16     cad aluno;          // declaração de aluno
17     aluno = Entrada(); // a estrutura aluno recebe o retorno da função Entrada
18     // saída
19     system("cls");
20     cout << "\nMatrícula do aluno: " << aluno.matricula << "CR: " << aluno.CR;
21     cout << "\n";
22     system("pause");
23     return(0);
24 }
25
26 cad Entrada()
27 {
28     cad temp;
29     cout << "\nMatrícula do aluno: ";
30     cin >> temp.matricula;
31     cout << "\nDigite CR: ";
32     cin >> temp.CR;
33     return(temp); // retorna uma estrutura
34 }
```

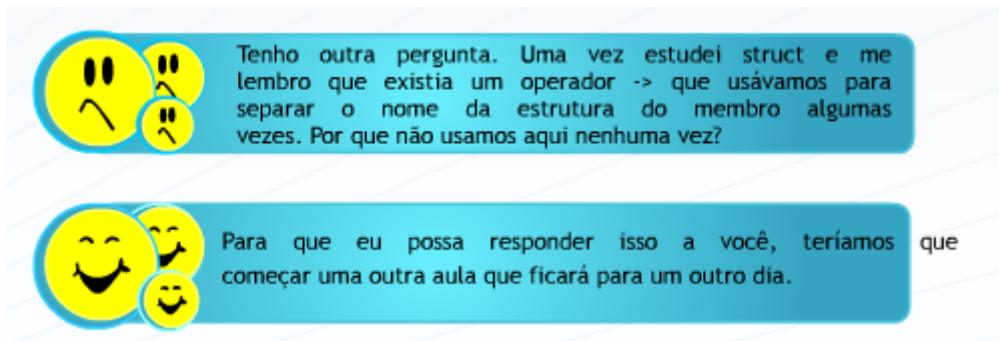
12 Funções como membros de estrutura

Neste exemplo, apresentaremos uma estrutura cujos dois membros são funções. Uma com retorno e outra, sem retorno.

A função
<pre> struct manipulaNumeros { int contaAlgarismos(int num) { int cont=0; while(num>0) { cont++; num/=10; } return cont; } void quocresto(int a, int b) { int aux; if(a<b) {aux=a; a=b; b=aux;} cout<<"\nQuociente: "<<a/b; cout<<"\nResto: "<<a%b; } }; </pre>
As chamadas das funções
<pre> cout<<"\nTotal de Algarismos: "<<num.contaAlgarismos(n1); num.quocresto(n1,n2); </pre>

funcMainEstudos.cpp
<pre> 1 #include <iostream> 2 using namespace std; 3 4 struct manipulaNumeros 5 { 6 int contaAlgarismos(int num) 7 { 8 int cont=0; 9 while(num>0) 10 { 11 cont++; 12 num/=10; 13 } 14 return cont; 15 } 16 17 void quocresto(int a, int b) 18 { 19 int aux; 20 if(a<b) {aux=a; a=b; b=aux;} 21 cout<<"\nQuociente: "<<a/b; 22 cout<<"\nResto: "<<a%b; 23 } 24 } 25 int main() 26 { 27 int n1,n2; 28 manipulaNumeros num; 29 cout<<"\nDigite um numero inteiro: "; cin>>n1; 30 cout<<"\nTotal de Algarismos: "<<num.contaAlgarismos(n1); 31 cout<<"\nDigite um numero inteiro: "; cin>>n2; 32 cout<<"\nDigite outro numero inteiro: "; cin>>n2; 33 num.quocresto(n1,n2); 34 cout<<"\n"; 35 system("pause"); 36 } </pre>

A screenshot of a Windows command-line window titled 'C:\Documents and Settings\Unita Lopes\Desktop...'. The window contains the following text:
Digite um numero inteiro: 3215467
Total de Algarismos: 7
Digite um numero inteiro: 34
Digite outro numero inteiro: 22
Quociente: 1
Resto: 12
Press any key to continue . . .



O que vem na próxima aula

Na próxima aula, você estudará os seguintes assuntos:

- Métodos de ordenação em arrays unidimensionais e em arrays de estruturas;
- Métodos de pesquisa em arrays unidimensionais e em arrays de estruturas.

CONCLUSÃO

Nesta aula, você:

- Compreendeu a importância das estruturas heterogêneas, visto que elas possibilitaram agrupar dados de tipos diferentes;
- Aprendeu a diferença entre definir e declarar uma estrutura heterogênea;
- Construiu programas usando estruturas heterogêneas;
- Construiu funções que recebiam ou retornavam estruturas;
- Construiu estruturas que tinham funções como membros.