#### Estruturas Heterogêneas

**Introdução**

Uma **estrutura heterogênea**, ou **registro**, é uma estrutura de dados que permite agrupar diferentes tipos de dados formando um novo tipo de dados. Esta estrutura é bastante interessante pois permite ao programador definir uma variável para descrever ***entidades*** do mundo real agrupando numa única estrutura variáveis que descrevem os diversos atributos relevantes daquela entidade. Por exemplo, uma ficha pessoal pode agrupar o nome, a idade, o telefone e o endereço de uma pessoal. Por sua vez, o endereço pode ser composto por rua, número, bairro, cidade, estado e CEP.

As partes que compreendem o registro são chamadas de **campos**. A sintaxe de declaração de um registro é a seguinte:

1. **Português Estruturado**:

O ***Visualg*** não implementa a declaração e uso de **registros**. Entretanto para algoritmos construídos em Português Estruturado, pode-se adotar a seguinte sintaxe:

NomeDoRegistro: **Registro**

Declaração de Campos

**FimRegistro**

Exemplo: para declarar a ficha pessoal citada acima:

TipoEndereco: **Registro**

rua[50]: caractere

numero: inteiro

bairro[30]: caractere

cidade[30]: caractere

UF[2]: caractere

CEP: inteiro

**FimRegistro**

TipoFicha: **Registro**

nome[50]: caractere

idade: inteiro

telefone[12]: caractere

endereco: TipoEndereco

**FimRegistro**

Deste modo, poderiam ter variáveis declaradas com estes novos tipos, criados pelo usuário. Por exemplo, poderia ser declarada a variável:

ficha: TipoFicha

Esta variável *ficha* teria então os campos *nome*, *idade*, *telefone* e *endereco*. Por sua vez, *endereco* seria um **campo composto**, isto é, contendo todos os campos declarados no registro TipoEndereco.

Para manipular uma variável do tipo Registro, na verdade o programador manipula cada campo do registro individualmente. A sintaxe para tal é: *NomeDoRegistro.NomeDoCampo*. Para campos compostos, a sintaxe é: *NomeDoRegistro.NomeDoCampoComposto.NomeDoCampo.* Por exemplo, para iniciar a variável *ficha*, declarada acima, poderia-se ter instruções tais como:

ficha.nome 🡨 “José da Silva”

ficha.idade 🡨 23

ficha.telefone 🡨 “(61)99123-4567”

ficha.endereco.rua 🡨 “das Farmácias”

ficha.endereco.numero 🡨 123

ficha.endereco.bairro 🡨 “Asa Sul”

ficha.endereco.cidade 🡨 “Brasília”

ficha.endereco.UF 🡨 “DF”

ficha.endereco.CEP 🡨 70000000

1. **Linguagem C**:

Em C, *registros* são definidos por meio da palavra reservada ***struct***. A sintaxe de declaração de registro em C é:

***struct*** nome\_do\_tipo\_do\_registro {

tipo1 campo1;

tipo2 campo2;

…

tipon campon;

};

Os registros declarados acima em Português Estruturado seriam “*traduzidos*” para C:

***struct*** TipoEndereco {

char rua[50];

int numero;

char bairro[30];

char cidade[30].

char UF[2];

long int CEP;

};

***struct*** TipoFicha {

char nome[50];

int idade;

char telefone[12];

struct TipoEndereco endereco;

};

A declaração da variável *ficha* seria em C:

***struct*** TipoFicha ficha;

E a inicialização das variáveis:

strcpy(ficha.nome, “José da Silva”);

ficha.idade = 23;

strcpy(ficha.telefone, “(61)99123-4567”);

strcpy(ficha.endereco.rua, “das Farmácias”);

ficha.endereco.numero = 123;

strcpy(ficha.endereco.bairro, “Asa Sul”);

strcpy(ficha.endereco.cidade, “Brasília”);

strcpy(ficha.endereco.UF, “DF”);

ficha.endereco.CEP = 70000000;

**Nota**: ***strcpy*** é uma função da linguagem C que copia uma cadeia de caracteres (2º parâmetro) para uma variável do tipo string (1º parâmetro). Assim, strcpy(ficha.nome, “José da Silva”); copia para o campo ***nome*** a cadeia “*José da Silva*”. Isso porque, por ser uma *string* na verdade um *vetor de char*, em C não pode haver uma atribuição direta da cadeia para o vetor, exceto no momento de sua criação!

1. **Linguagens Orientadas a Objeto**

Nas linguagens OO como Jave e Python, as ***entidades*** do mundo real são descritas por meio de ***classes***, dispensando o uso de registro. Classes incluem, além dos atributos que descrevem as entidades relacionadas (equivalentes aos registros), também os métodos, que modificam os estados destes atributos. Deste modo, aqui não entraremos em detalhes na descrição de classes, que serão melhor estudadas em disciplinas futuras.

**Vetores de Registros**

Vetores de registros são estruturas bem interessantes, pois definem na memória principal, uma estrutura equivalente às relações ou tabelas dos bancos de dados. Esta estrutura é adequada para armazenar uma coleção de objetos do mesmo tipo.

Para declarar um vetor de registros, primeiro declara-se o registro, E então, declara-se o vetor desse tipo.

Exemplo de declaração em Português Estruturado – será declarado um vetor com 40 elementos do tipo base Aluno, um registro contendo matrícula, nome e nota do aluno:

TipoAluno: **Registro**

matricula: inteiro

nome[50]: caractere

nota: real

**FimRegistro**

alunos[40]: TipoAluno

Esta estrutura vai produzir uma tabela de alunos que pode ser abstraída na figura a seguir:

**alunos**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **matricula** | **nome** | **nota** |
| 0 |  |  |  |
| 1 |  |  |  |
| 2 |  |  |  |
| . |  |  |  |
| . |  |  |  |
| . |  |  |  |
| 38 |  |  |  |
| 39 |  |  |  |

Para acessar um registro específico do vetor, deve-se indexar o nome do registro. Exemplo:

alunos[0].matricula 🡨 123

alunos[0].nome 🡨 “Maria das Graças”

alunos[0].nota 🡨 10.0

Essas atribuições produzem no vetor alunos:

**alunos**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **matricula** | **nome** | **nota** |
| 0 | 123 | Maria das Graças | 10.0 |
| 1 |  |  |  |
| 2 |  |  |  |
| . |  |  |  |
| . |  |  |  |
| . |  |  |  |
| 38 |  |  |  |
| 39 |  |  |  |

As declarações equivalentes em Linguagem C:

***struct*** TipoAluno {

int matricula

char nome[50];

float nota;

};

TipoAluno alunos[40];

alunos[0].matricula = 123;

strcpy(alunos[0].nome, “Maria das Graças”);

alunos[0].nota = 10.0;

**Problemas a serem resolvidos**

1. Faça um algoritmo que declare um registro *pessoa* com os campos nome (string de 20) e idade (inteiro). O algoritmo deve ter uma função que receba como parâmetros uma string de 20 e um int e atribua a um registro de pessoa estes dados e retorne ao chamador o registro atualizado; e outra função que receba um registro pessoa como parâmetro de entrada e mostra na tela os seus dados. O programa principal deve criar duas variáveis do tipo pessoa, a primeira variável deve ser atribuida com os valores “João” e 20 e a segunda variável deve ser atribuída com valores obtidos pelo teclado. Ambas devem ser iniciadas pela primeira função especificada. Depois a segunda função especificada deve ser chamada para mostrar os dados dos dois registros.

Solução em Português Estruturado:

**Algoritmo** “RegPessoa”

TipoPessoa: **Registro**

nome: string[20]

idade: inteiro

**FimRegistro**

**Funcao** iniciaPessoa(n: string, i: inteiro): TipoPessoa

**Var** p: TipoPessoa

**Inicio**

p.nome 🡨 n

p.idade 🡨 i

retorne p

**FimFuncao**

**Funcao** mostraPessoa(p: TipoPessoa)

**Inicio**

EscrevaL(p.nome, p.idade)

**FimFuncao**

**Var**

nome: string[20]

idade: inteiro

pessoa1, pessoa2: TipoPessoa

**Inicio**

pessoa1.nome 🡨 “João”

pessoa1.idade 🡨 20

Escreva(“Informe nome da 2ª pessoa: “)

Leia(nome)

Escreva(“Informe idade da 2ª pessoa: “)

Leia(idade)

Pessoa2 🡨 iniciaPessoa(nome, idade)

mostraPessoa(pessoa1)

mostraPessoa(pessoa2)

**FimAlgoritmo**

1. O algoritmo da questão anterior deve ser modificado para criar um vetor de 4 pessoas. No programa principal deve ser criado um *loop* para carregar todo o vetor de pessoas. E criado outro *loop* para mostrar todos os dados carregados no *loop* anterior.

Solução em Português Estruturado:

**Algoritmo** “RegPessoa”

**Constante** tamanho = 4

TipoPessoa: **Registro**

nome: string[20]

idade: inteiro

**FimRegistro**

**Funcao** iniciaPessoa(n: string, i: inteiro): TipoPessoa

**Var** p: TipoPessoa

**Inicio**

p.nome 🡨 n

p.idade 🡨 i

retorne p

**FimFuncao**

**Funcao** mostraPessoa(p: TipoPessoa)

**Inicio**

EscrevaL(p.nome, p.idade)

**FimFuncao**

**Var**

nome: string[20]

i, idade: inteiro

pessoas[tamanho] : TipoPessoa

**Inicio**

**Para** i **de** 0 **ate** tamanho – 1 **faca**

Escreva(“Informe nome da pessoa: “)

Leia(nome)

Escreva(“Informe idade da pessoa: “)

Leia(idade)

pessoas[i] 🡨 iniciaPessoa(nome, idade)

**FimPara**

**Para** i **de** 0 **ate** tamanho – 1 **faca**

mostraPessoa(pessoas[i])

**FimPara**

**FimAlgoritmo**

1. Faça um algoritmo com um registro Ponto, contendo apenas as coordenadas x e y de real. Crie uma função que receba como parâmentro dois pontos e retorne a distância entre ambos(utilize a formula de Pitágoras, para isso: d2 = (x1 – x2)2 + (y1 – y2)2 . No bloco principal, declare dois pontos e chame a função para calcular a distância entre ambos e mostre o resultado na tela.

Solução em Português Estruturado:

**Algoritmo** “RegPonto”

TipoPonto: **Registro**

x, y : real

**FimRegistro**

**Funcao** distancia(p1, p2: TipoPonto): real

**Var** d: real

**Inicio**

d 🡨 ((p1.x – p2.x)\*\*2 + (p1.y – p2.y)\*\*2)\*\*(1/2)

retorne d

**FimFuncao**

**Var**

ponto1, ponto2: TipoPonto

dist: real

**Inicio**

Escreva(“Informe a coordenada do primeiro ponto: “)

Leia(ponto1.x)

Leia(ponto1.y)

Escreva(“Informe a coordenada do segundo ponto: “)

Leia(ponto2.x)

Leia(ponto2.y)

dist 🡨 distancia(ponto1, ponto2)

EscrevaL(“Distância entre ponto1 e ponto2: “, dist)

**FimAlgoritmo**

1. Faça um algoritmo que declare um registro *produto*, com os campos codigo (inteiro) nome (string 30) e preco (real). No bloco principal do algoritmo, crie um vetor com 5 registros do produto. Os registros devem ser iniciados respectivamente com os dados:

3241 “Pé de moleque” R$ 0,75

3124 “Cocada baiana” R$ 0,60

1273 “Maria mole” R$ 0,45

2315 “Cajuzinho” R$ 0,80

3286 “Chocolate-barra” R$ 1,50

Ao final, todos os produtos devem ser mostrados na tela.

Solução em Português Estruturado:

**Algoritmo** “RegProdutos”

**Constante** tamanho = 5

TipoProduto: **Registro**

codigo: inteiro

nome: string[20]

preco: real

**FimRegistro**

**Funcao** iniciaProduto(c: inteiro, n: string, p: real): TipoProduto

**Var** prod: TipoProduto

**Inicio**

prod.codigo 🡨 c

prod.nome 🡨 n

prod.preco 🡨 p

retorne prod

**FimFuncao**

**Funcao** mostraProduto(p: TipoProduto)

**Inicio**

EscrevaL(“Cod. “,p.codigo, “, nome: “, p.nome, “, preço R$ “, p.preco)

**FimFuncao**

**Var**

i : inteiro

produtos[tamanho] : TipoProduto

**Inicio**

produtos[0] 🡨 iniciaProduto(3241, “Pé de moleque”, 0.75)

produtos[1] 🡨 iniciaProduto(3124, “Cocada baiana”, 0.60)

produtos[2] 🡨 iniciaProduto(1273, “Maria mole”, 0.45)

produtos[3] 🡨 iniciaProduto(2315, “Cajuzinho”, 0.80)

produtos[4] 🡨 iniciaProduto(3286, “Chocolate-barra”, 1.50)

**Para** i **de** 0 **ate** tamanho – 1 **faca**

mostraProduto(produtos[i])

**FimPara**

**FimAlgoritmo**

Solução em código C:

*// Autor: Vicente Teixeira*

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <locale.h>

#define tam 5

typedef struct produto {

int codigo;

char nome[20];

float preco;

}Produto;

*// PROTÓTIPOS DAS FUNÇÕES DA APLICAÇÃO*

Produto iniciaProduto (int c, char n[], float p);

void mostraProduto (Produto p);

int main() {

setlocale(LC\_ALL, "portuguese");

Produto produtos[tam];

produtos[0] = iniciaProduto(3241, "Pé de moleque", 0.75);

produtos[1] = iniciaProduto(3124, "Cocada baiana", 0.60);

produtos[2] = iniciaProduto(1273, "Maria mole", 0.45);

produtos[3] = iniciaProduto(2315, "Cajuzinho", 0.80);

produtos[4] = iniciaProduto(3286, "Chocolate-barra", 1.50);

for(int i = 0; i < tam; i++)

mostraProduto(produtos[i]);

printf("\n\n");

} *//main*

Produto iniciaProduto (int c, char n[], float p) {

Produto prod;

prod.codigo = c;

strcpy(prod.nome, n);

prod.preco = p;

return prod;

} *// iniciaProduto*

void mostraProduto (Produto p){

printf ("\n Cod.: %d, \tnome prod: %15s,\tpreço: R$ %.2f.", p.codigo, p.nome, p.preco);

} *// mostraProduto*