

PCI- Registros

Profa. Mercedes Gonzales
Márquez

Conceito

Registros, variáveis compostas heterogêneas (não homogêneas) ou estruturas são variáveis que podem conter uma coleção de objetos (potencialmente) de tipos diferentes.

- cada componente de uma estrutura é chamado de *membro* ou *campo*;
- cada campo está associado a um tipo de dado;

Sintaxe

```
struct <identificador estrutura> {  
  <tipo 1> <identificador campo 1>;  
  <tipo 2> <identificador campo 2>;  
  .....  
  <tipo k> <identificador campo k>;  
};
```

E na declaração fazemos:

```
struct <identificador estrutura> <nomevar1>, <nomevar2>, ...;
```

Sintaxe

Ou, podemos usar o typedef para redefinir o tipo

```
typedef struct{  
<tipo 1> <identificador campo1>;  
<tipo 2> <identificador campo2>;  
.....  
<tipo k> <identificador campok>;  
}<identificador estrutura>;
```

E na declaração usamos apenas:

```
<identificador estrutura> <nomevar1>,<nomevar2>, ...;
```

Exemplos de declarações

```
struct ponto2d {  
int x;  
int y;  
}
```

} Definição de tipo

```
struct ponto2d p1, p2;
```

} Declaração de variáveis p1 e p2
ou alternativamente pode ser também

```
typedef struct{  
int x;  
int y;  
}ponto2d;
```

} Definição de tipo

```
ponto2d p1,p2;
```

} Declaração de variáveis p1 e p2

Acesso aos campos de uma estrutura

- Para acessar os campos de uma estrutura usa-se o nome da variável do tipo struct e o nome do campo que se quer acessar, separando-os com . (ponto).

– No exemplo anterior:

p1.x e p1.y acessam os campos coordenada x e coordenada y respectivamente.

- Exemplo

```
ponto2d p1, p2;
```

```
p1.x = 0.0;
```

```
p1.y = 0.0;
```

```
p2.x = p1.x + 2;
```

```
p2.y = p1.y + 3;
```

Lendo e escrevendo registros

- A leitura dos campos de um registro deve ser feita campo a campo, como se fossem variáveis independentes.
- O mesmo vale para a escrita, que deve ser feita campo a campo.

Exercício:

Faça um programa que leia um ponto do plano cartesiano e escreva em qual quadrante ele se encontra.

Lendo e escrevendo registros

```
#include <stdio.h>
#include <stdlib.h>
typedef struct{
    int x;
    int y;
}ponto2d;
int main(){
    ponto2d p;
    scanf ("%d %d", &p.x, &p.y);
    if (p.x>0 && p.y>0)
        printf("o ponto (%d,%d) esta no 1o
quadrante", p.x,p.y);
    else if (p.x<0 && p.y>0)
        printf("ponto (%d,%d) esta no 2o
quadrante", p.x,p.y);
    else if (p.x<0 && p.y<0)
        printf("ponto (%d,%d) esta no 3o
quadrante", p.x,p.y);
    else if (p.x>0 && p.y<0)
        printf("ponto (%d,%d) esta no 4o
quadrante", p.x,p.y);
}
```


Vetor de Registros

- Podemos declarar um vetor de registros quando necessitamos de varias instâncias de um mesmo tipo de registro, por exemplo, para cadastrar todos os alunos de uma mesma turma.

- Como declarar um vetor de registros:

```
struct <identificador estrutura> nome_do_vetor[MAX];
```

- Como acessar um campo de registro de um vetor:

```
nome_do_vetor[indice].campo
```

Exemplo:

Faça um programa que permita o ingresso de 10 pontos do plano cartesiano e que escreva em qual quadrante eles se encontram.

Acesso aos campos de uma estrutura

```
#include <stdio.h>
#include <stdlib.h>
typedef struct{
    int x;
    int y;
}ponto2d;
int main(){
    ponto2d p[10];
    for (i=0; i<10; i++){
        scanf ("%d %d", &p[i].x, &p[i].y);
        if (p[i].x>0 && p[i].y>0)
            printf("Ponto no 1o quadrante");
        else if (p[i].x<0 && p[i].y>0)
            printf("Ponto no 2o quadrante");
        else if (p[i].x<0 && p[i].y<0)
            printf("Ponto no 3o quadrante");
        else if (p[i].x>0 && p[i].y<0)
            printf("Ponto no quarto
quadrante");
        }
    }
```

Exemplos de Estruturas

Exemplo :Cria uma estrutura para armazenar o nome, numero de matricula, 3 notas e a media de um aluno e leia esta informação para 10 alunos.

```
typedef struct /*Inicio da definição da estrutura */{  
    char nome[8];  
    int nmat; /* Número da matrícula */  
    float nota[3]; /* Notas */  
    float media; /* Média */  
}Cadastro; /* Fim da definição */
```

Exemplos de Estruturas

```
int main(){
Cadastro aluno[10];/* Declara uma
variável do tipo Cadastro */
int i;
printf (“Informe os dados de 10
alunos\n”);
for (i=0; i<10; i++){
printf (“nome do aluno\n”);
scanf (“%s”, aluno[i].nome);
printf (“numero de matricula\n”);
scanf (“%d”, &aluno[i].nmat);
printf (“Informe as tres notas\n”);
for (j=0;j<3; j++)
scanf (“%f”,&aluno[i].nota[j]);
aluno[i].media= (aluno[i].nota[0] +
aluno[i].nota[1]
aluno[i].nota[2])/3.0;
}
```

```
/* Imprimindo*/
printf (“Imprimindo dados dos 10
alunos\n”);
for (i=0; i<10; i++){
printf (“aluno %d\n”, i+1);
printf (“nome:
%s\n”,aluno[i].nome);
printf (“numero de matricula:
%d\n”,aluno[i].nmat );
printf (“Nota 1: %2.2f - Nota 2:
%2.2f – Nota 3:%2.2f\n”,
aluno[i].nota[0], aluno[i].nota[1],
aluno[i].nota[2]);
printf (“media das notas:
%2.2f\n”,aluno[i].media);
}system("PAUSE");
}
```

Aninhamento de Estruturas

- Pode-se também declarar um registro como uma das variáveis de um registro, quantas vezes isso for necessário.
- Exemplo:

```
typedef struct {  
    int dia;  
    int mes;  
    int ano;  
}data;
```

```
typedef struct {  
    char nome[10];  
    int cargo;  
    data data_nasc;  
} Funcionario;
```

Aninhamento de Estruturas

```
Funcionario func1;  
func1.data_nasc.dia = 01;  
func1.data_nasc.mes = 10;  
func1.data_nasc.ano = 2001;
```

Funções com registros como parâmetros

- Uma função não somente considera dados primitivos como parâmetros, podemos também ter necessidade da passagem de dados tipo registro como no exemplo abaixo.
Ex. Calcular e escrever a área total de 10 tetraedros, dadas as coordenadas de cada um de seus quatro vértices. Para tanto, deverão ser utilizados os seguintes módulos:
 - (a) Que calcula a distância entre dois pontos do espaço;
 - (b) Que calcula a área de um triângulo em função de seus lados
$$AREA = \sqrt{\rho \times (\rho - a) \times (\rho - b) \times (\rho - c)}$$

onde ρ é o semi-perímetro do triângulo $(a+b+c)/2$.

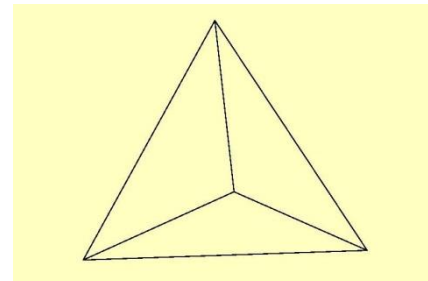
Funções com registros como parâmetros

- Como os 4 vértices do tetraedro são pontos no espaço, podemos criar um registro chamado ponto3d, que considere as três coordenadas de um ponto no espaço.

Assim:

```
typedef struct{  
    float x,y,z;  
}Ponto3d;
```

- Assim podemos usar um vetor de 4 elementos do tipo ponto3d que contenha os 4 vértices do tetraedro.



Funções que retornam um registro

No seguinte exemplo temos uma função que recebe as coordenadas x , y e z por separado e devolve uma única variável tipo estrutura Ponto3d com os dados recebidos*/

```
Ponto3d cria_pt (float x, float y, float z) {  
    Ponto3d tmp;  
    tmp.x=x;  
    tmp.y=y;  
    tmp.z=z;  
    return tmp;  
}
```

Ponteiros para registros

- Ao criarmos uma variável de um tipo struct, esta é armazenada na memória como qualquer outra variável, e portanto possui um endereço.
- Exemplo

```
#include <stdio.h>
```

```
typedef struct {
```

```
    double x;
```

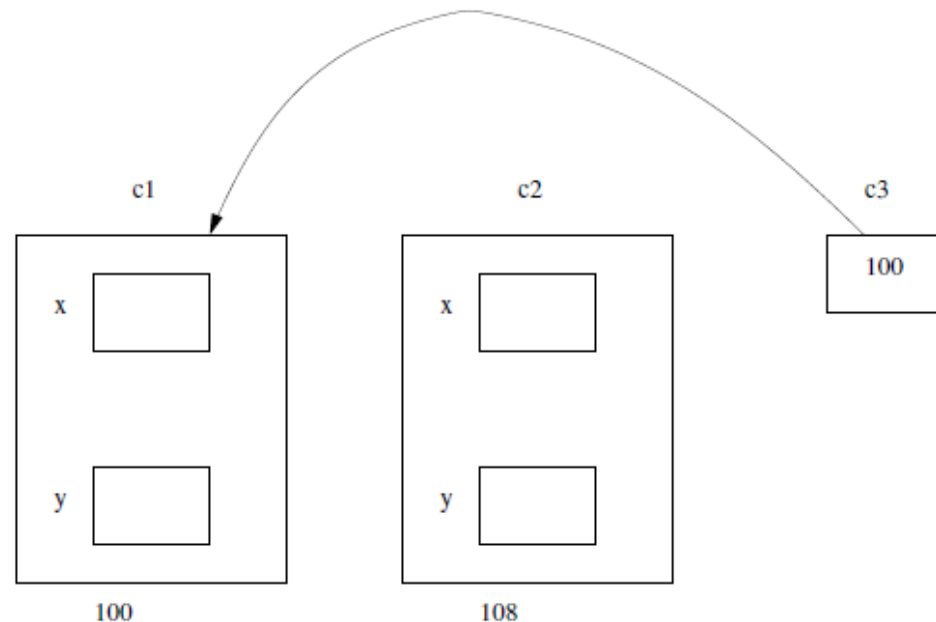
```
    double y;
```

```
} ponto2d;
```

```
int main(){
```

```
    ponto2d c1, c2, *c3;
```

```
    c3 = &c1;
```



Ponteiros para registros

- Para acessarmos os campos de uma variavel struct via um ponteiro, podemos utilizar o operador * juntamente com o operador . como de costume:

```
ponto2d c1, *c3;
```

```
c3 = &c1;
```

```
(*c3).x = 1.5;
```

```
(*c3).y = 1.5;
```

- Em C também podemos usar o operador -> para acessar campos de uma estrutura via um ponteiro. Podemos obter o mesmo resultado do exemplo anterior:

```
ponto2d c1, *c3;
```

```
c3 = &c1;
```

```
c3->x = 1.5; c3->y = 1.5;
```

Ponteiros para registros

- Resumindo: Para acessar campos de estruturas via ponteiros use um dos dois:
- `ponteiroEstrutura->campo`
- `(*ponteiroEstrutura).campo`

Exercício 1:

- Vamos criar as seguinte funções:
- `void lePonto3D(ponto3d *f);`: lê dados de uma ponto no espaço tridimensional com coordenadas (x,y,z) passada como ponteiro. (Por quê como ponteiro?)
- `void imprimePonto3D(ponto3d f);` Imprime coordenadas de um ponto.
- O programa principal que requeira a leitura de 5 pontos no espaço tridimensional.

Ponteiros para registros

- ponto3d ProdutoEscalar(ponto3d v1,ponto3d v2)

onde produto_escalar= $\langle v1, v2 \rangle = v1.x * v2.x + v1.y * v2.y$