Introdução a Programação

Tipos Estruturados de Dados

Tópicos da Aula

- Hoje aprenderemos a trabalhar com tipos de dados mais complexos
  - Tipos Primitivos x Tipos Estruturados
  - Conceito de Tipos Estruturados
  - Importância
  - Tipos Estruturados em C (struct)
  - Declaração de structs
  - Variáveis do tipo struct
  - Operações com estruturas
  - Comando typedef
  - Passando Estruturas para Funções
  - Estruturas Aninhadas

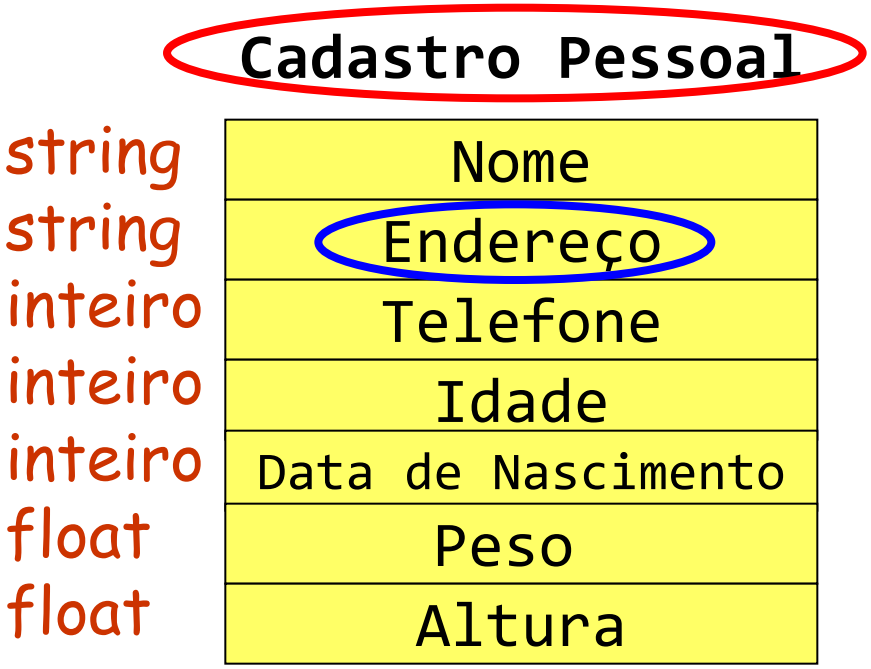
Tipos Estruturados

- C oferece tipos primitivos que servem para representar valores simples
  - Reais (float, double), inteiros (int), caracter (char)
- C oferece também mecanismos para estruturar dados complexos nos quais as informações são compostas por diversos campos

Tipos Estruturados !

Tipos Estruturados

- Agrupa conjunto de tipos de dados distintos sob um único nome
- Podemos criar varios objetos na memória de um determinado tipo estruturado
  - Estruturas ou Registros



Nome do tipo estruturado

Membro do tipo estruturado

Tipos Estruturados em C (struct)

- Forma Geral:

```
struct   nome_do_tipo  {
 declaração de variável 1 ;
 declaração de variável n ;
};
```

Cadastro Pessoal

Nome
Endereço
Telefone
Idade
Data de Nascimento
Peso
Altura

```
struct cadastro_pessoal {
        char  nome[50];
        char  endereço[100];
        int       telefone;
        int       idade;
        int       nascimento;
        float   peso;
        float   altura;
};
```

Importância de Tipos Estruturados

- Considere um ponto representado por duas coordenadas: x e y
- **Sem** mecanismos para agrupar as duas coordenadas:

```
int main() {

float   x ;

float   y ;

…

}
```

Não dá para saber que estas variáveis representam coordenadas de um ponto

Importância de Tipos Estruturados

- Uma estrutura em C serve para agrupar diversas variáveis dentro de um único contexto

```
struct   ponto     {
      float   x ;
      float   y ;
};
```

Declarando Variáveis do Tipo Ponto

- A estrutura *ponto* passa a ser um tipo
- Então, podemos declarar uma variável deste tipo da seguinte forma:

```
struct   ponto     {
      float    x ;
      float    y ;
};
int main()  {
    struct    ponto    p ;
     ...
}
```

A variável é do tipo `struct ponto`

Acessando Membros do Tipo Ponto

- Membros de uma estrutura são acessados via o operador de acesso (".")
  - Para acessar as coordenadas:

```
struct   ponto     {
      float    x ;
      float    y ;
};
int main() {
    struct    ponto    p ;
    p.x = 0.0;
    p.y = 7.5;
    ...
}
```

O nome da variável do tipo `struct ponto` deve vir antes do "."

Utilizando oTipo Estruturado Ponto

```
*/ programa que captura e imprime coordenadas*/
#include <stdio.h>
struct   ponto  {
      float    x ;
      float    y ;
} ;

int  main (){
   struct   ponto   p ;
   printf("\nDigite as coordenadas do ponto (x,y)") ;
   scanf ("%f %f", &p.x , &p.y ) ;
   printf("O ponto fornecido foi:(%f,%f)\n",p.x, p.y);
   return   0 ;
}
```

variável p não precisa de parênteses

Onde Declarar um Tipo Estruturado?

- Geralmente, declara-se um tipo estruturado fora das funções
  - Escopo da declaração engloba todas as funções no mesmo arquivo fonte
- Pode-se, também, declarar tipos estruturados dentro de funções
  - Neste caso, escopo do tipo estruturado é na função

Declarando oTipo Estruturado Fora das Funções

```
struct   ponto  {
    float    x ;
    float    y ;
} ;
```

Declarado fora das funções

```
struct ponto alteraPonto(float x, float y){
   struct ponto q;
   q.x = x;
   q.y = y;
   return q;
}
int  main (){
   struct   ponto   p ;
   printf("\nDigite as coordenadas do ponto (x,y)") ;
   scanf ("%f %f", &p.x , &p.y ) ;
   p = alteraPonto(8,9);
   ...
}
```

Funções podem usar o tipo estruturado declarado acima

Declarando oTipo Estruturado Dentro de uma Função

```
void leCoordenadas(float* x, float* y){
    struct   ponto  {
       float     x ;
       float     y ;
    };
    struct ponto q;
    scanf ("%f %f", &q.x , &q.y ) ;
    *x = q.x;
    *y = q.y;
}
int  main (){
    struct   ponto   p ;
    printf("\nDigite as coordenadas do ponto (x,y)") ;
    leCoordenadas(&p.x , &p.y ) ;
    ...
}
```

Declarado dentro da função alteraPonto

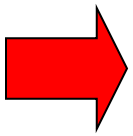
Outra função NÃO enxerga a declaração do tipo estruturado

Errado !

Outras Formas de Declarar Tipos Estruturados e Variáveis

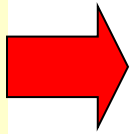
- Pode-se declarar um tipo estruturado e uma variável deste tipo de diferentes formas

```
struct   ponto     {
     float    x ;
     float    y ;
} p;
```



Declara-se tipo e variável numa expressão só

```
struct {
     float    x ;
     float    y ;
} p;
```



Declara-se tipo SEM NOME e variável numa expressão só

Cuidado com legibilidade !

Inicializando Variáveis de Tipos Estruturados

- Uma variável de um tipo estruturado pode ser inicializada com uma estrutura com o auxílio do abre-fecha parênteses("**{**" e "**}**")

```
struct   pessoa     {
      char  nome[60] ;
      int   idade ;
};
int main() {
    struct   pessoa   p = {"Ana", 30};
    ...
}
```

Inicialização da variável `p` do tipo `struct pessoa`

Deve-se inicializar os membros na ordem correta

Atribuição de Estruturas

- A estrutura armazenada por uma variável de um tipo estruturado pode ser atribuída a outra variável deste mesmo tipo estruturado

```
struct   pessoa     {
      char   nome[60] ;
      int    idade ;
};
int main() {
    struct    pessoa    p1,p2;
    p1 = {"Ana", 30};

    p2 = p1;
    ...
}
```

Atribuição de uma estrutura para a variável p1 (Errado)

Atribuição da estrutura contida em p1 para p2

Outras Operações com Estruturas

Como escrever um programa que imprime a soma das coordenadas de dois pontos?

```
struct   ponto    {
      float x ;
      float y ;
};
int main() {
    struct   ponto   p3;
    struct ponto p1 = {0.0, 4.5};
    struct ponto p2 = {1.0,2.5};
    p3 = p1 + p2;
    printf("O x e y do novo ponto é:%f,%f,p3.x,p3.y);
    return 0;
}
```

Não podemos somar estruturas inteiras

Errado !

Outras Operações com Estruturas

Como escrever um programa que imprime a soma das coordenadas de dois pontos?

```
struct   ponto    {
      float x ;
      float y ;
};
int main() {
    struct   ponto   p3;
    struct ponto p1 = {0.0, 4.5};
    struct ponto p2 = {1.0,2.5};
    p3.x = p1.x + p2.x;
    p3.y = p1.y + p2.y;
    printf("O x e y do novo ponto é:%f,%f",p3.x,p3.y);
    return 0;
}
```

Temos que somar membro a membro

Certo !

Usando *typedef*

- O comando **`typedef`** permite criar novos nomes para tipos existentes
  - Criação de sinônimos para os nomes de tipos
  - É útil para abreviar nomes de tipos ou tipos complexos
- Forma Geral:

```
typedef tipo_existente sinonimo;
```

Usando *typedef*

- Após a definição de novos nomes para os tipos, pode-se declarar variáveis usando estes nomes

```
struct   pessoa     {
      char   nome[60] ;
      int    idade ;
};
typedef struct pessoa Pessoa;
int main() {
   Pessoa    p = {"Ana", 30};

    ...
}
```

Tipo existente

Novo nome

Simplificou declaração do tipo de variável

Usando *typedef*

- Podemos combinar **`typedef`** com declaração do tipo estruturado

```
typedef struct   pessoa     {
      char  nome[60] ;
      int   idade ;
} Pessoa;
int main() {
    Pessoa    p = {"Ana", 30};

    ...
}
```

Criação de tipo e criação de sinônimo

Passagem de Estruturas para Funções

- Considere a função abaixo:

```
void   imprimePonto  ( struct  ponto   p ){
   printf("O ponto fornecido foi:(%f,%f)\n",p.x,p.y);
}
```

  - Assim como podemos passar tipos primitivos como argumentos para uma função, podemos passar estruturas

Passagem de Estruturas para Funções

```
*/ programa que captura e imprime coordenadas*/
#include <stdio.h>
typedef struct   ponto  {
      float    x ;
      float    y ;
} Ponto;

void imprimePonto(Ponto q) {
   printf("O ponto fornecido foi:(%f,%f)\n",q.x, q.y);

int  main (){
   Ponto   p ;
   printf("\nDigite as coordenadas do ponto (x,y)") ;
   scanf ("%f %f", &p.x , &p.y ) ;
   imprimePonto(p);
   return   0 ;
}
```

Passa a estrutura armazenada em p como argumento

Retornando Estruturas

- Considere a função abaixo:

```
struct ponto alteraPonto(float x, float y ){
    struct ponto q;
    q.x = x;
    q.y = y;
    return q;
}
```

- Assim como uma função pode retornar um valor de um tipos primitivo, uma função pode retornar uma estrutura

Retornando Estruturas

```
typedef struct   ponto  {
    float    x ;
    float    y ;
} Ponto;

Ponto alteraPonto(float x, float y){
   Ponto q;
   q.x = x;
   q.y = y;
   return q;
}
int  main (){
   Ponto   p ;
   printf("\nDigite as coordenadas do ponto (x,y)") ;
   scanf ("%f %f", &p.x , &p.y ) ;
   p = alteraPonto(8,9);
   ...
}
```

Variável p recebe a estrutura retornada por alteraPonto

Tipos Estruturados Mais Complexos

- Aninhamento de estruturas
  - Membros de uma estrutura podem ser outras estruturas previamente definidas
  - Exemplo:

```
typedef struct   ponto  {
      float    x ;
      float    y ;
} Ponto;
```

```
typedef struct circulo {
      Ponto centro;
      float raio;
} Circulo;
```

Tipo estruturado **Circulo** tem como um dos membros um **Ponto**

Usando os Tipos Ponto e Circulo

Função que calcula a distância entre 2 pontos

```
float  distancia ( Ponto  p ,  Ponto  q ){
   float d = sqrt((q.x - p.x )*(q.x - p.x)+
                  (q.y - p.y)*(q.y - p.y)) ;
   return  d ;
}
```

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Função que determina se um ponto está no círculo

```
int   interior  ( Circulo   c ,   Ponto  p ){
   float   d  =  distancia ( c.centro ,  p ) ;
   return  ( d <= c.raio ) ;
}
```

Passa para distancia uma estrutura Ponto que é membro da estrutura Circulo

Resumindo …

- Tipos Estruturados
- Structs
- Operações com Estruturas
- Comando typedef
- Passando Estruturas para Funções
- Estruturas Aninhadas