Pontifícia Universidade Católica
do Rio de Janeiro

INF 1007 – Programação II

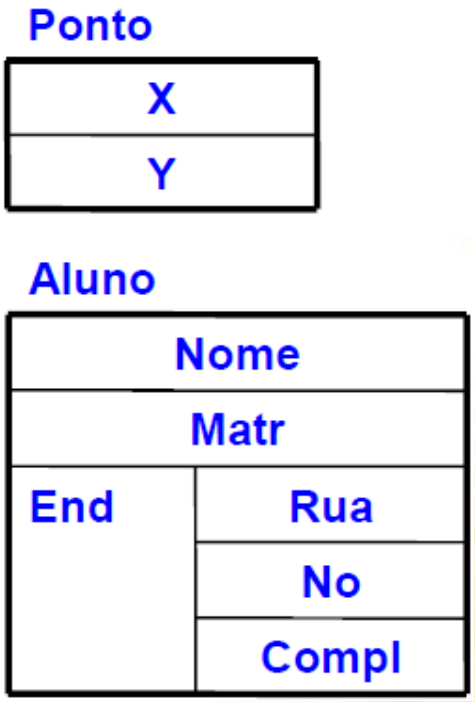
Aula 06 – Tipos Estruturados

Edirlei Soares de Lima
<elima@inf.puc-rio.br>

visionlab
visualization laboratory

Dados Compostos

- Até agora somente utilizamos **tipos de dados simples**: `char, int, float, double`.
- Muitas vezes precisamos manipular **dados compostos ou estruturados**.
- **Exemplos:**
  - pontos no espaço bidimensional
    - representado por duas coordenadas (x e y), mas tratado como um único objeto (ou tipo)
  - registros de alunos;
    - aluno representado pelo seu nome, número de matrícula, endereço, etc ., estruturados em um único objeto (ou tipo)



Tipo Estrutura – `Struct`

- Tipo estrutura (`struct`):
  - tipo de dado com campos compostos de tipos mais simples
  - elementos acessados através do operador de acesso “ponto” (.)

```
struct ponto      /* declaração da estrutura ponto */
{
  float x;
  float y;
};

int main(void)
{
  struct ponto p; /* declara p como variável do tipo struct ponto */
  p.x = 10.0;     /* acessa os elementos de ponto */
  p.y = 5.0;
  printf("%f %f", p.x, p.y);
}
```

Tipo Estrutura – Exemplo

```
/* Captura e imprime as coordenadas de um ponto qualquer */
#include <stdio.h>

struct ponto
{
  float x;
  float y;
};

int main (void)
{
  struct ponto p;
  printf("Digite as coordenadas do ponto(x y): ");
  scanf("%f %f", &p.x, &p.y);
  printf("O ponto fornecido foi: (%.2f, %.2f)\n", p.x, p.y);
  return 0;
}
```

Ponteiro de Estruturas

- É possível utilizar ponteiros para estruturas:

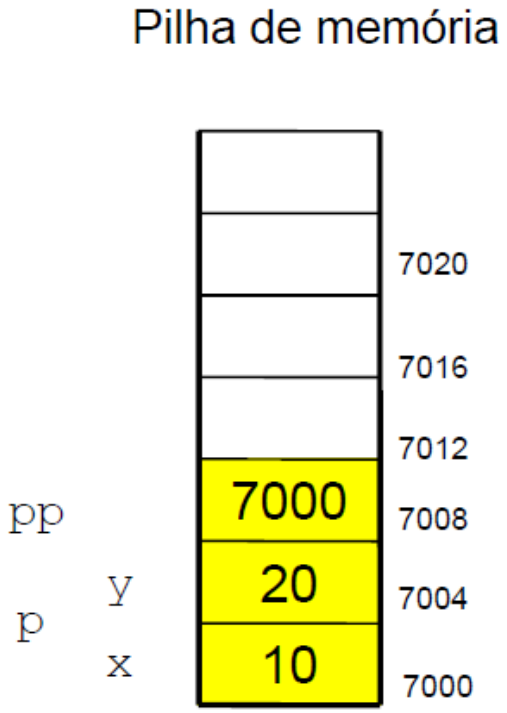
```
struct ponto p;
struct ponto *pp=&p;
...
(*pp).x = 12.0;
pp->x = 12.0;
p.x = 12.0;
(&p)->x =12.0;
```

formas equivalentes de acessar o valor de um campo x

- Ponteiros para estruturas:
  - acesso ao valor de um campo x de uma variável estrutura p: `p.x`
  - acesso ao valor de um campo x de uma variável ponteiro pp: `pp->x`
  - acesso ao endereço do campo x de uma variável ponteiro pp: `&pp->x`

Ponteiro de Estruturas

```
struct ponto
{
  float x;
  float y;
};
int main(void)
{
  struct ponto p = {10, 20};
  struct ponto *pp;
  pp = &p;
}
```



- Qual o valor de... ?

~~pp.x~~ (&p)->x &(p.y)

p.y pp->x &(pp->y)

Passagem de Estruturas por Valor para Funções

- A passagem de estruturas para funções funciona de forma semelhante a passagem de variáveis simples:

```
/* função que imprime as coordenadas do ponto */
void imprime(struct ponto p)
{
  printf("O ponto fornecido foi: (%.2f,%.2f)\n", p.x, p.y);
}
```

- A função recebe toda a estrutura como parâmetro:
  - função acessa a cópia da estrutura na pilha
  - função não altera os valores dos campos da estrutura original
  - operação pode ser custosa se a estrutura for muito grande

Estruturas – Passagem por Referência

- Assim como acontece com variáveis simples, também é possível passar estruturas por referencia:

```
/* função que imprima as coordenadas do ponto */
void imprime(struct ponto* pp)
{
  printf("O ponto fornecido foi: (%f, %f)\n", pp->x, pp->y);
}
```

- A função recebe o endereço da estrutura como parâmetro:
  - a função pode alterar os valores dos campos da estrutura original
  - operação menos custosa do que a passagem por valor

Estruturas – Passagem por Referência

```
void imprime(struct ponto* pp)
{
  printf("O ponto fornecido foi: (%f, %f)\n", pp->x, pp->y);
}

void captura(struct ponto* pp)
{
  printf("Digite as coordenadas do ponto(x y): ");
  scanf("%f %f", &pp->x, &pp->y);
}

int main(void)
{
  struct ponto p;
  captura(&p);
  imprime(&p);
  return 0;
}
```

Alocação Dinâmica de Estruturas

- É possível alocar estruturas dinamicamente:
  - o tamanho do espaço de memória alocado dinamicamente é dado pelo operador sizeof aplicado sobre o tipo estrutura;
  - a função malloc retorna o endereço do espaço alocado, que é então convertido para o tipo ponteiro da estrutura.

```
struct ponto *p;
p = (struct ponto*)malloc(sizeof(struct ponto));

p->x = 12.0;

free(p);
```

Definição de Novos Tipos

- A linguagem C permite a definição de nomes para novos tipos de dados através do comando `typedef`:
- Exemplo:

```
typedef int* PInt;
typedef float Vetor[4];

Vetor v;    /* exemplo de declaração usando Vetor */
v[0] = 3;
```

  - `PInt` : um tipo ponteiro para int;
  - `Vetor` : um tipo que representa um vetor de quatro elementos;

- Útil para abreviar nomes de tipos e para tratar tipos complexos

Definição de Novos Tipos

- É possível utilizar o comando `typedef` para definir o nome de um tipo estruturado:

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

Representa a estrutura ponto

Representa o tipo ponteiro para a estrutura ponto

- Simplifica a utilização da estrutura ponto:

```
Ponto p1;
PPonto p2;
p1.x = 10.0;
p2 = (PPonto)malloc(sizeof(Ponto));
p2->x = 5.0;
```

Definição de Novos Tipos

- É possível combinar o comando `typedef` com a declaração da estrutura:

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

Estruturas Aninhadas

- Os campos de uma estrutura podem ser outras estruturas
  - Exemplo: Definição de um círculo composto por um ponto central e um raio.

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

struct circulo
{
  Ponto p; /* centro do círculo */
  float r; /* raio do círculo */
};
typedef struct circulo Circulo;
```

Estruturas Aninhadas

- Função para a calcular distância entre 2 pontos:

```
float distancia(Ponto* p, Ponto* q)
{
  float d = sqrt(pow(q->x - p->x, 2) + pow(q->y - p->y, 2));
  return d;
}
```

- Função para determinar se um ponto está ou não dentro de um círculo:

```
int interior (Circulo* c, Ponto* p)
{
  float d = distancia(&c->p, p);
  if (d < c->r)
    return 1;
  else
    return 0;
}
```

Estruturas Aninhadas

- Função principal:

```
int main(void)
{
  Circulo c;
  Ponto p;
  printf("Digite as coordenadas do centro e o raio do circulo:\n");
  scanf("%f %f %f", &c.p.x, &c.p.y, &c.r);
  printf("Digite as coordenadas do ponto:\n");
  scanf("%f %f", &p.x, &p.y);
  if (interior(&c,&p) == 1)
    printf("Pertence ao interior!\n");
  else
    printf("Não pertence ao interior!\n");
  return 0;
}
```

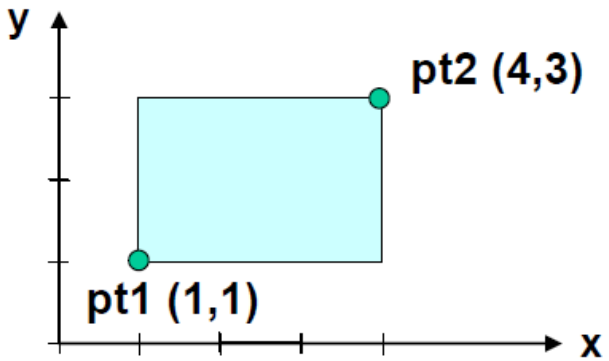
Estruturas Aninhadas

- Estrutura de um retângulo:

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

struct retangulo
{
  Ponto pt1;
  Ponto pt2;
};

typedef struct retangulo Retangulo;
```




```
Retangulo meu_retangulo;
meu_retangulo.pt1.x = 1.0;
meu_retangulo.pt1.y = 1.0;
meu_retangulo.pt2.x = 4.0;
meu_retangulo.pt2.y = 3.0;
```

Funções para Criação de Estruturas

- Função para criar a estrutura de um Ponto:

```
Ponto criaPonto(int x, int y)
{
  Ponto p;
  p.x = x;
  p.y = y;
  return p;
}
```

- Função para criar dinamicamente a estrutura de um Retângulo:

```
Ponto* criaRetangulo(Ponto p1, Ponto p2)
{
  Retangulo *rect = (Retangulo*)malloc(sizeof(Retangulo));
  rect->pt1 = p1;
  rect->pt2 = p2;
  return rect;
}
```

Funções para Criação de Estruturas

- Função Principal:

```
int main (void)
{
  Retangulo *r1;

  r1 = criaRetangulo(criaPonto(10,10), criaPonto(20,25));

  printf("P1: %d %d", r1->pt1.x, r1->pt1.y);
  printf("P2: %d %d", r1->pt2.x, r1->pt2.y);

  free(r1);

  return 0;
}
```

Vetores de Estruturas

- **É possível criar vetores de estruturas;**

- **Exemplo:** criar uma função para calcular o centro geométrico de conjunto de pontos:
  - entrada: vetor de estruturas definindo o conjunto de pontos;
  - saída: centro geométrico, dado por:

$$\bar{x} = \frac{\sum x_i}{n} \qquad \bar{y} = \frac{\sum y_i}{n}$$

Vetores de Estruturas

```
Ponto centro_geometrico(int n, Ponto *v)
{
  int i;
  Ponto p = {0, 0};
  for (i=0; i<n; i++)
  {
    p.x += v[i].x;
    p.y += v[i].y;
  }
  p.x = p.x/n;
  p.y = p.y/n;
  return p;
}
```

v é um vetor de estruturas Ponto

Função retornando estrutura:

- para estruturas pequenas, este recurso facilita o uso da função
- para estruturas grandes, a cópia do valor de retorno pode ser cara

Vetores de Estruturas

```
int main (void)
{
  Ponto vet_pontos[3] = {{10,15}, {50, 30}, {20, 30}};
  Ponto centro;

  centro = centro_geometrico(3, vet_pontos);

  printf("Centro Geometrico: %f, %f\n", centro.x, centro.y);

  return 0;
}
```

Vetor de estrutura inicializado estaticamente.

Vetores de Estruturas – Usando Alocação Dinâmica

```
Ponto *centro_geometrico(int n, Ponto* v)
{
  int i;
  Ponto *p = (Ponto*)malloc(sizeof(Ponto));
  p->x = 0;
  p->y = 0;
  for (i=0; i<n; i++)
  {
    p->x += v[i].x;
    p->y += v[i].y;
  }
  p->x = p->x/n;
  p->y = p->y/n;
  return p;
}
```

Função retornando ponteiro para estrutura.

Vetores de Estruturas – Usando Alocação Dinâmica

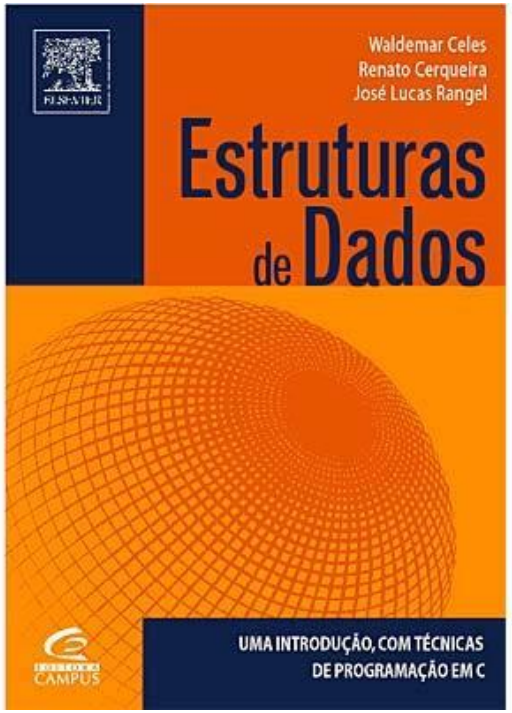
```
int main (void)
{
  int x;
  Ponto *vet_pontos;
  Ponto *centro;
  vet_pontos = (Ponto*)malloc(3 * sizeof(Ponto));
  for (x = 0; x < 3; x++)
  {
    printf("Digite o ponto %d: ", x+1);
    scanf("%f %f", &vet_pontos[x].x, &vet_pontos[x].y);
  }
  centro = centro_geometrico(3, vet_pontos);
  printf("Centro Geometrico: %f, %f\n", centro->x, centro->y);
  free(centro);
  free(vet_pontos);
  return 0;
}
```

Vetor de estrutura inicializado dinamicamente.

Leitura Complementar

- Waldemar Celes, Renato Cerqueira, José Lucas Rangel, **Introdução a Estruturas de Dados**, Editora Campus (2004).

- **Capítulo 8 – Tipos Estruturados**



Exercícios

Lista de Exercícios 04 – Tipos Estruturados

http://www.inf.puc-rio.br/~elima/prog2/