# Programação 2

***Tipos Estruturados***

Rivera

# Tipo Estrutura

- Motivação
  - Manipulação de dados compostos ou estruturados
  - Exemplos
    - Ponto no espaço bidimensional
      - P = (x, y)
    - Dados associados a aluno
      - Aluno
        - » Nome
        - » Matrícula
        - » Endereço
          - ➢ Rua
          - ➢ Numero
          - ➢ Complemento

estrutura é definida com a motivação de manipulação de dados em conjunto, com uma definida organização.

# Tipo Estrutura

- Tipo estrutura
  - Tipo de dados com campos compostos
  - Acesso a elementos através de “ponto” ( . )

```
struct ponto
{
    float x;
    float y;
};
...
int main ( void )
{
    struct ponto  p;    // declara  p como variável tipo struct ponto
    ...
    p.x = 10.0;
    p.y = 5.0 + p.x;
    ...
    printf (“ O ponto P = (%f, %f)”, p.x, p.y);
    return 0;
}
```

elementos nomeados dentro de uma estrutura podem ser acessados por meio do prefixo "." ou seja, p.x acessa o elemento x dentro da estrutura p.

# Tipo Estrutura

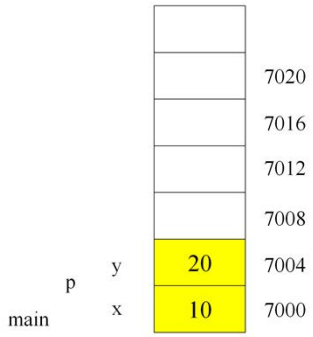
- Ponteiros para estruturas
  - Acesso ao valor do campo x da variável estrutura p: p.x
  - Acesso ao valor do campo x da variável ponteiro pp: p->x
  - Acesso ao endereço do campo x da variável ponteiro pp: &pp->x

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

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

estruturas podem também ser utilizadas com ponteiros, utilizando de diferentes prefixos para que seja possível o acesso a dados, como -> e &->

# Qual o valor de ...?

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

int main ( )
{
    struct ponto p;
    struct ponto *pp;
    pp = &p
    ...
}
```




pp->x

&(pp->y)

p.y

&(p.y)

(&p)->x

pp.x

p.y == 20
pp.x == 7000
&(p.y) == 7004
pp->x == 10
&(pp->y) == 7004
(&p)->x == 7000

# Passagem de estruturas por valor para funções

- Análoga à passagem de variáveis simples
- 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 grande

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

Void imprime (struct ponto p)
{
    printf (“O ponto fornecido foi: (%.2f, %.2f) \n”, p.x, p.y);
}
```

é possível o uso de estruturas como parâmetros para funções, realizando ações mantendo sua organização.

# Estruturas como valor de retorno

```
#include <stdio.h>

struct ponto { float x, y; };

struct ponto le(void)
{
   struct ponto tmp;
   printf (“Digite xoordenadas (x, y): ”);
   scanf (“%f %f”, &tmp.x, &tmp.y);
   return tmp;
}

int main (void)
{
   struct ponto p = le( );
   printf(“ O ponto fornecido foi: (%.2f, %.2f) \n”, p.x, p.y);
   return 0;
}
```

estruturas também podem ser acessadas para retornar dados internos individualmente

# Passagem de estruturas por referência para função

- Apenas o ponteiro da estrutura é passada

```
void imprime (struct ponto *pp)
{
   printf(“ O ponto fornecido foi: (%.2f, %.2f)\n”, pp->x, pp->y);
}

void captura (struct ponto *pp)
{
   printf(“ Digite o ponto (x, y): ”);
   scantf(“%f %f”, &pp->x, &pp->y);
}

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

é possível também o uso de ponteiros como refêrencia de estruturas, para funções, onde apenas o ponteiro é passado, sem manipular a estrutura em si.

## Alocação dinâmica de estruturas

- Tamanho do espaço de memória alocado dinamicamente é dado pelo operador sizeof aplicado sobre o tipo estrutura
- 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);
```

é possível alocar dinâmicamente uma estrutura, adequando-se o espaço reservado a seu tamanho, e retornando o endereço do espaço alocado como ponteiro para a estrutura

# Definição de Novos Tipos

- `typedef`
  - Permite criar nomes de tipos
  - Útil para abreviar nomes de tipos e para tratar tipos

```
typedef unsigned char Uchar;
typedef int* Pint;
typedef float V4f[4];

Uchar j;
V4f v;
...
v[0] 3.5;
...
```

- Uchar : tipo char sem sinal
- Pint : tipo ponteiro para int
- V4f : tipo representado um vetor de 4 float

é possível criar novos nomes de tipos, facilitando manipulações e nomenclaturas, utilizando a função typedef

# Definição de Novos Tipos

- `typedef`
  - Definição de nomes de tipo estruturas

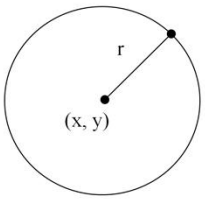
```
struct ponto {
  float x, y;
};

typedef struct ponto Pto;
typedef struct ponto *Ppto;
```

```
typedef struct ponto {
  float x, y;
} Pto;
```

**Exemplo :**

Definição da estrutura de um círculo

```
struct circulo {
  Pto p;
  float r;
};

typedef struct circulo Circulo;
```




é possível também, criar um novo tipo, de tipo de estrutura já existente, como por exemplo o tipo Pto, que é do tipo struct ponto, que contém os valores x, y floats.

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

typedef struct ponto{
   float x, y;
} Pto;

typedef struct circulo{
   Pto p;
   float r;
} Circulo;

void main (void)
{
   Circulo c;
   Pto p;
   int w;
   scanf("%f %f %f", __ , ___ , ___);  // circulo
   scanf("%f %f", ___, ___ );   // ponto
   w = interior(____, _____);    // circulo e ponto
   printf("\n Pertence ao interior: %d\n", w);
   return 0;
}
```

```c
// verifica se um ponto está no interior do círculo

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

int interior (Circulo *c, Pto* p)
{
   float d = distancia(&c->p, p);
   return (d < c->r);
}
```

# Vetores de estruturas

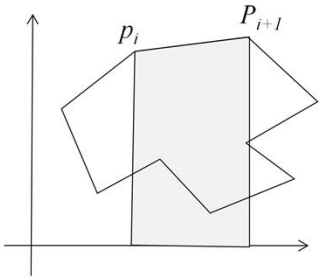
```
// centro geométrico de n pontos

 Pto centroGeometrico (int n, Ponto* v)
 {
    int i;
    Pto p = {0.0f, 0.0f};    // inicializa ponto
    for (i=0; i<n; i++)
    {
       p.x += v[i].x;
       p.y += v[i].y;
    }
    p.x /= n;
    p.y /= n;
    return p;
 }
```

- Função retornando estrutura:
  - Para estruturas pequenas:
    - Facilita o uso da função
  - Para estruturas grandes:
    - A copia de valores é cara.

retornar estruturas é uma operação custosa, porém facilita o uso em caso de estruturas pequenas. Em caso de estruturas grandes, é melhor retornar de maneira individual os dados desejados, prezando o desempenho.

# Vetores de estruturas

- Exemplo
  - Função que calcula área de um polígono plano definido por uma sequencia de n pontos
    - Área: soma das áreas dos trapézios formado pelos lados do polígono e o eixo x
    - Área do trapézio: projeção sobre eixo x da aresta $p_i$ a $p_{i+1}$, dada por
      - $a_i = (x_{i+1} - x_i)\,(y_{i+1} + y_i) / 2$
    - Algumas áreas são negativas
    - Áreas externas ao polígono são anuladas



# Vetores de estruturas

```
// area de um polígono

 float areaPolig (int n, Ponto* p)
 {









 }
```

## Vetores de Ponteiros para Estruturas

- Exerc.
  - Escrever um programa
    - Gerar um vetor de n círculos
      - Centro (x, y) e raio r gerados aleatoreamente ou lidos de um arquivo.
    - Gerar um vetor de m pontos (x, y)
    - Verificar quais círculos contem cada ponto (x, y)
    - Criar um polígono (regular) tendo como vértices os centros dos círculos