

# Estruturas e Enumeradores

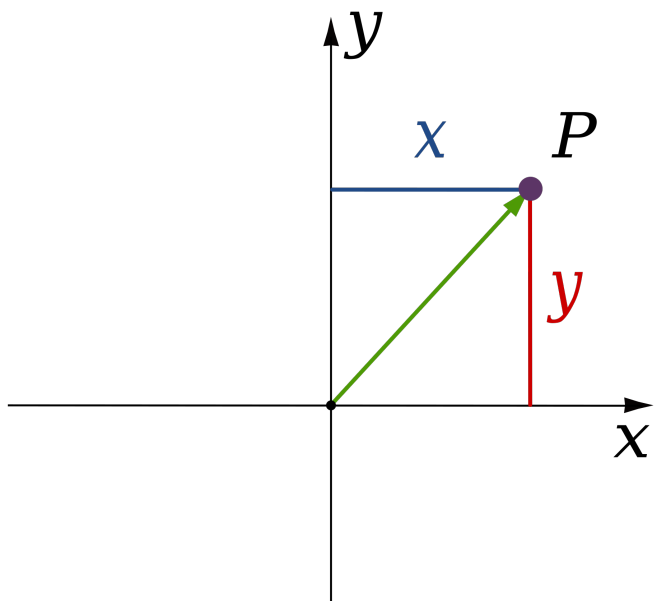
---

Programação e Desenvolvimento de Software I  
Heitor Ramos

# Problema 1

Você foi contratado para fazer um software gráfico que trabalha com figuras geométricas. Como representar um ponto, um círculo e um triângulo?

# Ponto



- Um ponto pode ser representado como um valor em 2 eixos
- Vamos pensar em algumas operações comuns de um ponto

- Rotação

$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta.$$

- Translação

$$(x', y') = (x + a, y + b).$$

# Problema 1.1

- Vamos iniciar representando um ponto
- Criar 2 códigos para rotação e translação

# Código

## Representação

```
double x;
```

```
double y;
```

## Translação:

```
x = x + a; // x += a  
y = y + b; // y += b
```

## Rotação:

```
double x_antigo = x;  
double y_antigo = y;  
x = x_antigo * cos(theta) - y_antigo * sin(theta);  
y = x_antigo * sin(theta) + y_antigo * cos(theta);
```

# Outras Figuras

- Círculos

- 1 ponto
- Raio

- Triângulos

- 3 pontos (6 doubles!)

- Retângulos

- 1 ponto
- Base
- Altura

- Polígono simples

- n pontos!!

# Rotacionando Figuras + Complexas (Ex: triângulo)

```
double xa_antigo = xa;
```

```
double ya_antigo = ya;
```

```
double xb_antigo = xb;
```

```
double yb_antigo = yb;
```

```
double xc_antigo = xc;
```

```
double yc_antigo = yc;
```

Rapidamente o código vira uma bagunça (aumenta a chance de erro)

```
xa = xa_antigo * cos(theta) - ya_antigo * sin(theta);
```

```
ya = xa_antigo * sin(theta) + ya_antigo * cos(theta);
```

```
xb = xb_antigo * cos(theta) - yb_antigo * sin(theta);
```

```
yb = xb_antigo * sin(theta) + yb_antigo * cos(theta);
```

```
xc = xc_antigo * cos(theta) - yc_antigo * sin(theta);
```

```
yc = xc_antigo * sin(theta) + yc_antigo * cos(theta);
```



# Rotacionando Figuras + Complexas

```
double xa_antigo = xa;  
double ya_antigo = ya;  
  
double xb_antigo = xb;  
double yb_antigo = yb;  
  
double xc_antigo = xc;  
double yc_antigo = yc;
```

- Podemos usar vetores
- Vamos melhorar um pouco com structs

# Structs

- Já aprendemos sobre tipos simples (int, double, char, float)
- Agora vamos ver como combinar os mesmos
- Pensando em um ponto:
  - double coordenada x
  - double coordenada y
- Como representar um ponto?

```
struct ponto {  
    int x;  
    int y;  
};  
int main(void) {  
    int a = 72;  
    int b = 98;  
    struct ponto p;  
    p.x = 1;  
    p.y = 2;  
    struct ponto t  
    t.x = p.x + a;  
    t.y = p.y + b;  
}
```

# Entendendo a memória

```
struct ponto {  
    int x;  
    int y;  
};  
int main(void) {  
    → int a = 72;  
    int b = 98;  
    struct ponto p;  
    p.x = 1;  
    p.y = 2;  
    struct ponto t;  
    t.x = p.x + a;  
    t.y = p.y + a;  
}
```

nome	end(&)	val(*)
	0x0048	
	0x0044	
a	0x0040	72
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# Entendendo a memória

```
struct ponto {  
    int x;  
    int y;  
};  
int main(void) {  
    int a = 72;  
    → int b = 98;  
    struct ponto p;  
    p.x = 1;  
    p.y = 2;  
    struct ponto t;  
    t.x = p.x + a;  
    t.y = p.y + a;  
}
```

nome	end(&)	val(*)
	0x0048	
	0x0044	
a	0x0040	72
b	0x003c	98
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# Entendendo a memória

```
struct ponto {  
    int x;  
    int y;  
};  
int main(void) {  
    int a = 72;  
    int b = 98;  
    struct ponto p;  
    p.x = 1;  
    p.y = 2;  
    struct ponto t;  
    t.x = p.x + a;  
    t.y = p.y + a;  
}
```



nome	end(&)	val(*)
	0x0048	
	0x0044	
a	0x0040	72
b	0x003c	98
p.x	0x0038	1
p.y	0x0034	2
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# Entendendo a memória

```
struct ponto {  
    int x;  
    int y;  
};  
int main(void) {  
    int a = 72;  
    int b = 98;  
    struct ponto p;  
    p.x = 1;  
    p.y = 2;  
    struct ponto t  
    t.x = p.x + a;  
    t.y = p.y + a;  
}
```



nome	end(&)	val(*)
	0x0048	
	0x0044	
a	0x0040	72
b	0x003c	98
p.x	0x0038	1
p.y	0x0034	2
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# Entendendo a memória

```
struct ponto {  
    int x;  
    int y;  
};  
int main(void) {  
    int a = 72;  
    int b = 98;  
    struct ponto p;  
    p.x = 1;  
    p.y = 2;  
    struct ponto t  
    t.x = p.x + a;  
    t.y = p.y + a;  
}
```



nome	end(&)	val(*)
	0x0048	
	0x0044	
a	0x0040	72
b	0x003c	98
p.x	0x0038	1
p.y	0x0034	2
t.x	0x0030	73
t.y	0x002c	100
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# Structs não precisam ter campos do mesmo tipo

- Como representar um aluno?!
- Matricula (int)
- Nome (char)

# Structs com vários tipos

```
struct aluno {  
    char nome[20];  
    int  mat;  
};  
  
int main(void) {  
    struct aluno a;          // declara um aluno  
    strcpy(a.nome, "Fulano"); // seta o nome  
    a.mat = 20311028;        // seta a matricula  
  
    return 0;  
}
```

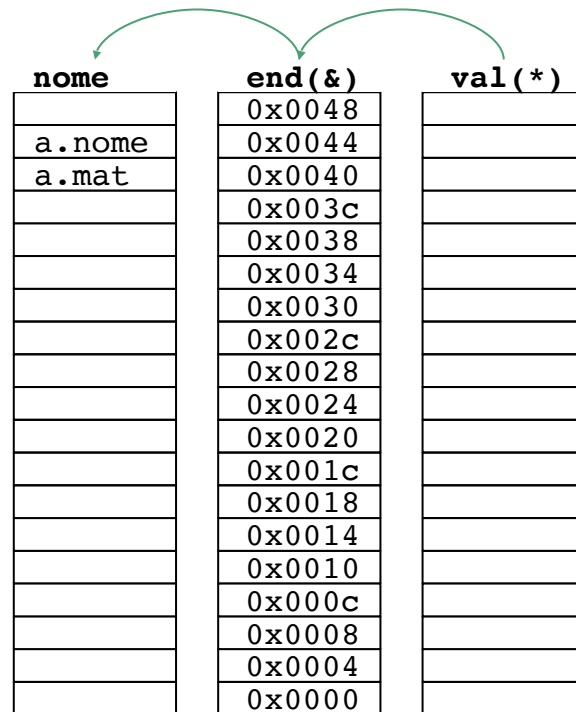
# Structs com vários tipos

```
struct aluno {  
    char nome[20];  
    int  mat;  
};
```

```
int main(void) {  
→ struct aluno a;
```

```
    strcpy(a.nome, "Flavio");  
    a.mat  = 20311028;
```

```
    return 0;  
}
```



# Structs com vários tipos

```
struct aluno {  
    char nome[20];  
    int  mat;  
};
```

```
int main(void) {  
    struct aluno a;
```

```
    strcpy(a.nome, "Flavio");
```

```
    → a.mat  = 20311028;
```

```
    return 0;
```

```
}
```

nome	end(&)	val(*)
	0x0048	
a.nome	0x0044	
a.mat	0x0040	
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	'\0'
	0x0014	'o'
	0x0010	'i'
	0x000c	'v'
	0x0008	'a'
	0x0004	'l'
	0x0000	'F'

# Structs com vários tipos

```
struct aluno {  
    char nome[20];  
    int  mat;  
};
```

```
int main(void) {  
    struct aluno a;
```

```
➔ strcpy(a.nome, "Flavio");  
  a.mat  = 20311028;
```

```
    return 0;
```

```
}
```

nome	end(&)	val(*)
	0x0048	
a.nome	0x0044	
a.mat	0x0040	
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
nome[6]	0x0018	'\0'
nome[5]	0x0014	'o'
nome[4]	0x0010	'i'
nome[3]	0x000c	'v'
nome[2]	0x0008	'a'
nome[1]	0x0004	'l'
nome[0]	0x0000	'F'

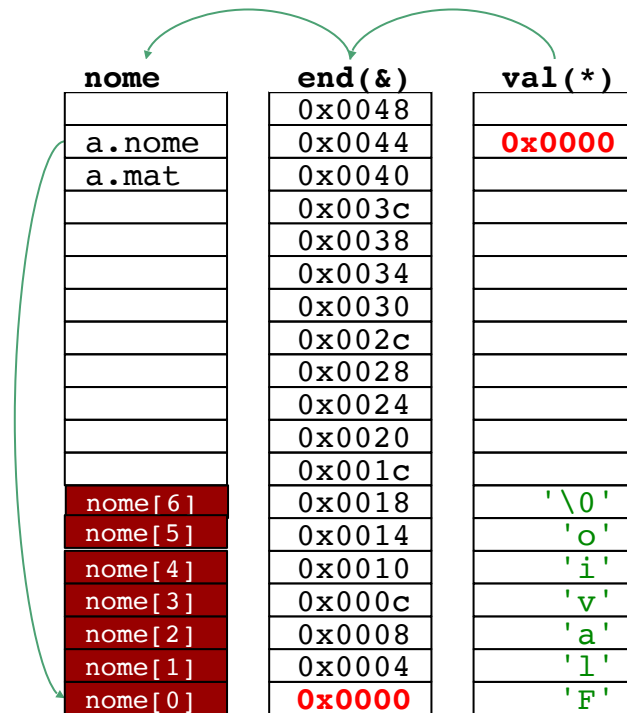
# Structs com vários tipos

```
struct aluno {  
    char nome[20];  
    int  mat;  
};
```

```
int main(void) {  
    struct aluno a;
```

```
→ strcpy(a.nome, "Flavio");  
  a.mat  = 20311028;
```

```
    return 0;  
}
```



# Structs com vários tipos

```
struct aluno {  
    char nome[20];  
    int mat;  
};
```

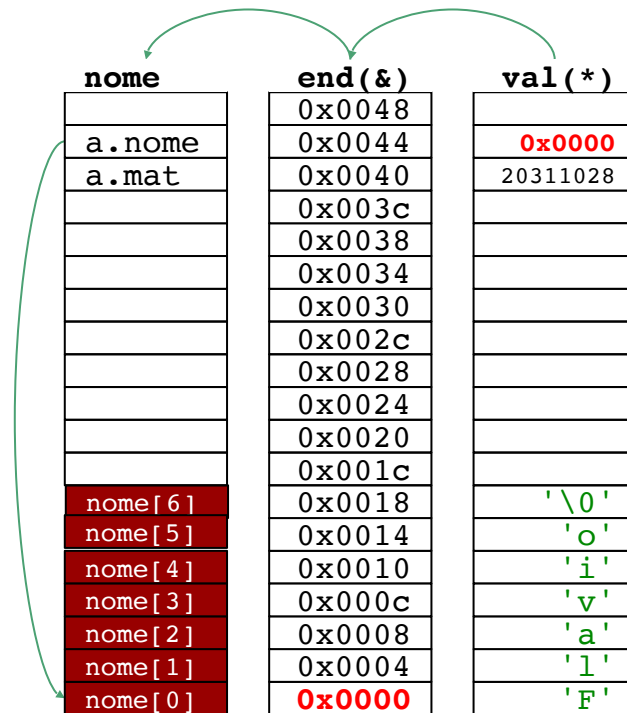
```
int main(void) {  
    struct aluno a;
```

```
    strcpy(a.nome, "Flavio");
```

```
    → a.mat = 20311028;
```

```
    return 0;
```

```
}
```



# Typedef

- **typedef** é uma palavra reservada para definir novos tipos
- `typedef double nota_t;`
  - Define um "alias"/pseudônimo de double
- Como um CPF pode extrapolar os limites de int, podemos definir os mesmos como um vetor de dígitos
- `typedef int cpf_t[11];`

# Typedef

- Cuidado com os typedefs
- Pense bem se faz sentido criar um tipo novo
  - Talvez faça para o CPF
  - Não para a nota
- Geralmente faz sentido com structs!

# Poupando Caracteres com typedef

```
typedef struct {  
    double x;  
    double y;  
} ponto_t;
```

```
int main(void) {  
    ponto_t p;  
  
    // ...  
    return 0;  
}
```

## Problema 2

Faça um programa que lê dois números do teclado e inicia um struct ponto

## Problema 2

```
typedef struct{
    double x;
    double y;
}ponto_t;
int main(void) {
    ponto_t ponto;
    printf("Digite a coordenada x: ");
    scanf("%lf", &ponto.x);
    printf("Digite a coordenada y: ");
    scanf("%lf", &(ponto).y);

    // ...
    return 0;
}
```

## Problema 3

Crie os tipos para círculo, retângulos, triângulos e algumas funções comuns de cada como: rotação, translação etc.

# Tipos Compostos

```
typedef struct {  
    double x;  
    double y;  
} ponto_t;
```

```
typedef struct {  
    ponto_t centro;  
    double raio;  
} circulo_t;
```

```
typedef struct {  
    ponto_t base;  
    double altura;  
    double largura;  
} retangulo_t;
```

```
// triângulo são 3 pontos  
typedef ponto_t triangulo_t[3];
```

# Acessando dados de tipos compostos

```
#include <stdio.h>
typedef struct
{
    double x;
    double y;
}ponto_t;
typedef struct
{
    ponto_t centro;
    double raio;
}circulo_t;
int main()
{
    circulo_t c;
    circulo_t c1 = {{1, 2}, 10};
    c.centro.x = 0.5;
    c.centro.y = 0.9;
    c.raio = 2.3;
}
```

# Array de estrutura

- O “.” Vem depois do []

```
#include <stdio.h>

typedef struct
{
    double x;
    double y;
}ponto_t;

typedef ponto_t triangulo_t[3];

int main()
{
    triangulo_t triangulo1;
    for (int i = 0; i < 3; i++)
    {
        printf("Digite x y do ponto%d: ",i+1);
        scanf("%lf %lf", &triangulo1[i].x, &triangulo1[i].y);
    }
}
```

# Atribuição entre estruturas

- Só pode ser feito se os campos forem iguais!
- No caso de arrays, atribuição de elementos é válida

```
#include <stdio.h>
#include <string.h>
typedef struct
{
    char nome[50];
    int idade;
    char rua[100];
}cadastro_t;
int main()
{
    cadastro_t pessoa1, pessoa2, pessoas[10];
    strcpy(pessoa1.nome, "Fulano");
    pessoa1.idade = 35;
    strcpy(pessoa1.rua, "Rua dos bobos, n. zero");
    strcpy(pessoas[1].nome, "Beltrano");
    pessoas[1].idade = 39;
    strcpy(pessoas[1].rua, "Rua das árvores, n. 1");
    pessoa2 = pessoa1;
    pessoas[2] = pessoas[1];
}
```

# Enumerações

- Definem um tipo com valores determinados em um certo domínio

```
#include <stdio.h>
enum colors {branco, azul, marrom, lilas};
typedef enum
{
    segunda=1, terca=2, quarta=3, quinta=4, sexta=5, sabado=0, domingo=0
}dia;

int main()
{
    enum colors cordapele = marrom;
    enum colors cordocabelo = lilas;
    dia hoje = segunda;
    dia amanha = hoje+1;
    dia noname = 15;

    if(hoje == 0)
        printf("Final de semana!\n");
    else
        printf(":-(");
}
```

# Alignment, Padding e Data Packing

```
#include <stdio.h>

// Alignment requirements
// (typical 32 bit machine)

// char      1 byte
// short int  2 bytes
// int        4 bytes
// double     8 bytes

// structure A
typedef struct structa_tag
{
    char      c;
    short int  s;
} structa_t;

// structure B
typedef struct structb_tag
{
    short int  s;
    char      c;
    int        i;
} structb_t;

// structure C
typedef struct structc_tag
{
    char      c;
    double    d;
    int        s;
} structc_t;

// structure D
typedef struct structd_tag
{
    double    d;
    int        s;
    char      c;
} structd_t;

int main()
{
    printf("sizeof(structa_t) = %lu\n", sizeof(structa_t));
    printf("sizeof(structb_t) = %lu\n", sizeof(structb_t));
    printf("sizeof(structc_t) = %lu\n", sizeof(structc_t));
    printf("sizeof(structd_t) = %lu\n", sizeof(structd_t));

    return 0;
}
```

# Union

```
#include <stdio.h>

union teste {
    int a;
    float b;
};

int main()
{
    union teste un01;

    int valor;
    un01.a = 15;
    un01.b = 20.50;

    printf("Tamanho Inteiro: %lu\n", sizeof(int));
    printf("Tamanho Float: %lu\n\n", sizeof(float));

    printf("Valor do membro a: %d\n", un01.a);
    printf("Valor do membro b: %f\n", un01.b);
    return 0;
}
```