

# *Linguagens de Programação 1*

**Francisco Sant'Anna**

**Sala 6020-B**

`francisco@ime.uerj.br`

`http://github.com/fsantanna-uerj/LP1`

# Tipos Compostos

# O que é um tipo?

- “Natureza” ou “Classificação” de um dado
- Número, Texto (string), Booleano
  - Número real, inteiro, inteiro de 1 byte, ...
- `v=100 ; type(v)`
- `v=100.0 ; type(v)`
- `int v = 100;`
- `float v = 100;`

# Para que tipos?

- Recusar operações inválidas
- Documentar o código
- Especializar por tamanho
- Desempenho

# Tipos “Básicos”

- Numéricos: **char**, **short**, **int**, **float**, *etc.*
- Ponteiros: *tipo\**
- Vetores/Arrays: *tipo[n]*
- **int** x;  
**int** xs[10];  
**int**\* p = xs;  
\*(p+3) = 100;

**structs**

# structs

- *Record, Registro, Product Type*
- Construtor de tipos novos
  - construtor **E**
  - *campo1 (de tipo1) E campo2 (de tipo2) E ...*
- Exemplo:
  - Um Personagem é representado por um tipo composto por  
`forca(int) E energia(int) E experiencia(int)`

# structs - Declaração

```
#include <stdio.h>

struct Personagem {
    int forca;
    int energia;
    int experiencia;
};

void main (void) {
    struct Personagem p1;
    p1.forca      = 10;
    p1.energia    = 100;
    p1.experiencia = 0;

    struct Personagem p2 = { 13, 150, 200 };

    printf("> %d %d\n", p1.forca, p2.forca);
}
```



# structs - Leitura

```
#include <stdio.h>

struct Personagem {
    int forca;
    int energia;
    int experiencia;
};

void main (void) {
    struct Personagem p1;

    int forca;
    scanf("%d", &forca);
    p1.forca = forca;

    scanf("%d", &p1.energia);

    printf("%d %d\n", p1.forca, p1.energia);
}
```

# structs - Ponteiros

```
#include <stdio.h>

struct Personagem {
    int forca;
    int energia;
    int experiencia;
};

void main (void) {
    struct Personagem p1;
    scanf("%d", &p1.forca);

    struct Personagem* ptr = &p1;

    printf("%d %d %d\n",
           p1.forca, (*ptr).forca, ptr->forca);
}
```

# structs - Leitura

```
#include <stdio.h>

struct Personagem {
    int forca;
    int energia;
    int experiencia;
};

// preencheA?, preencheB?

void main (void) {
    struct Personagem p1 = preencheA();

    struct Personagem p2;
    preencheB(&p2);

    printf("%d %d\n", p1.forca, p2.forca);
}
```

# structs - Leitura

```
#include <stdio.h>

struct Personagem {
    int forca;
    int energia;
    int experiencia;
};

struct Personagem preencheA (void) {
    ...
}

void main (void) {
    struct Personagem p1 = preencheA();
    printf("%d\n", p1.forca);
}
```

# structs - Leitura

```
#include <stdio.h>

struct Personagem {
    int forca;
    int energia;
    int experiencia;
};

void preencheB (struct Personagem* p) {
    ...
}

void main (void) {
    struct Personagem p2;
    preencheB(&p2);
    printf("%d\n", p2.forca);
}
```

**unions**

# unions

- União, Variante, *Sum Type*
- Construtor de tipos novos
  - construtor **OU**
  - *campo1 (de tipo1) OU campo2 (de tipo2) OU ...*
- Exemplo:
  - Uma Identidade é representada por um tipo composto por  
`IFP(int) OU CPF(int) OU NOME(char[255])`

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

union Ident {
    int ifp;
    int cpf;
    char nome[256];
};

int main (void) {
    union Ident i1;
    i1.ifp = 117766118;

    union Ident i2;
    i2.cpf = 1688833355;

    union Ident i3;
    strcpy(i3.nome, "Francisco Sant'Anna");

    printf("> %d %d %s\n", i1.ifp, i2.cpf, i3.nome);

    return 0;
}
```



# unions

- Como fica a memória?
- O que acontece se gravar como um subtipo e ler como outro?
- C possui tipagem fraca
  - Similar ao problema do tamanho de um vetor...
- Solução?

# União Discriminada

- Solução?
  - Guardar o subtipo em uso

```
struct Ident {  
    int sub; // [0,1,2]  
    union {  
        int ifp; // sub=0  
        int cpf; // sub=1  
        char nome[255]; // sub=2  
    };  
};
```

```
void exhibe (struct Ident* id) {  
    switch (id->sub) {  
        case 0:  
            printf("IFP = %d\n", id->ifp);  
            break;  
        case 1:  
            ...  
        case 2:  
            ...  
        default:  
            // erro  
    }  
}
```

# structs + unions

- Maneiras de declarar e usar
  - Separado
  - Aninhado
  - Aninhado / Separado

**Teste**

**Próxima Sexta 24/05**

**Conteúdo deste slide  
(struct e union)**