



STRUCT

ECM404

PROBLEMA

- ❑ Deseja-se armazenar o nome, idade e telefone de uma pessoa.

```
char nome[50];  
int idade;  
char telefone[15];
```

- ❑ E se aumentássemos para 4 pessoas?

```
char nome1[50], nome2[50], nome3[50], nome4[50];  
int idade1, idade2, idade3, idade4;  
char telefone1[15], telefone2[15], telefone3[15], telefone4[15];
```

- ❑ As variáveis vistas até agora podem ser classificadas em duas categorias:
 - ❑ **Simples:** definidas por tipos **int**, **float**, **double** e **char**;
 - ❑ **Compostas:** definidas por **array** de um mesmo tipo.
- ❑ No entanto, a linguagem C permite que se criem novas estruturas a partir de tipos básicos. Esta nova estrutura é chamada de **struct**.

STRUCT

- ❑ Uma nova estrutura pode ser vista como um **novo tipo de dado** formado pela composição de variáveis de outros tipos:
- ❑ Pode ser declarada em qualquer escopo (localmente ou globalmente);

```
struct nomeStruct{  
    tipo1 nomeVariavel_1;  
    tipo2 nomeVariavel_2;  
    ...  
    tipoN nomeVariavel_N;  
};
```

❑ Exemplo: Cadastro de pessoas

```
struct cadastro{  
    char nome[50];  
    int idade;  
    char telefone[15];  
};
```

char nome[50]

int idade

char telefone[15]

cadastro

- ❑ Como as informações pertencem a uma mesma pessoa, logo, faz sentido agrupá-las;
- ❑ Além disso, caso existam outras pessoas a serem cadastradas, o agrupamento facilitará o manuseio destes dados.

STRUCT

- ❑ Uma vez definida a estrutura, uma **variável** deste novo tipo pode ser declarado de modo similar aos tipos já existentes.

```
struct cadastro{  
    char nome[50];  
    int idade;  
    char telefone[15];  
};  
  
int main (int argc, char *argv[]){  
  
    struct cadastro c;  
  
    return 0;  
}
```

STRUCT

- ❑ Uma vez definida a estrutura, uma **variável** deste novo tipo pode ser declarado de modo similar aos tipos já existentes.

```
struct cadastro{
    char nome[50];
    int idade;
    char telefone[15];
};

int main (int argc, char *argv[]){

    struct cadastro c;

    return 0;
}
```

Obs.: por ser um tipo definido pelo programador, usa-se a palavra **struct** antes do tipo da nova variável;

PROBLEMA DO CADASTRO

❑ Voltando para o cadastro das 4 pessoas:

```
char nome1[50], nome2[50], nome3[50], nome4[50];  
int idade1, idade2, idade3, idade4;  
char telefone1[15], telefone2[15], telefone3[15], telefone4[15];
```

```
struct cadastro{  
    char nome[50];  
    int idade;  
    char telefone[15];  
};  
  
int main (int argc, char *argv[]){  
  
    struct cadastro c1, c2, c3, c4;  
  
    return 0;  
}
```


ACESSANDO AS VARIÁVEIS

- ❑ Para acessar as variáveis utilizaremos o operador **ponto** (.)

```
int main (int argc, char *argv[]){  
    struct cadastro c1, c2, c3, c4;  
  
    strcpy(c1.nome, "Fulano de Tal");  
  
    scanf("%i", &c1.idade); // ou c1.idade = 26;  
  
    strcpy(c1.telefone, "(011)98765-4321");  
  
    return 0;  
}
```

INICIALIZAÇÃO DE STRUCTS

- ❑ Assim como nos arrays, um struct pode ser previamente inicializado.

```
struct ponto{  
    int x;  
    int y;  
};  
  
int main (int argc, char *argv[]){  
    struct ponto p = {110,220};  
  
    return 0;  
}
```

PROBLEMA DO CADASTRO

- ❑ Voltando para o cadastro. E se aumentássemos para 100?
- ❑ Lembrando:
 - ❑ **struct:** define um novo tipo de dado que representa um conjunto de variáveis que podem ser de tipos diferentes;
 - ❑ **array:** é uma sequência de elementos de mesmo tipo.

PROBLEMA DO CADASTRO

- ❑ Podemos criar um array do tipo cadastro.

```
struct cadastro{
    char nome[50];
    int idade;
    char telefone[16];
};

int main (int argc, char *argv[]){

    struct cadastro listaCadastro[100];

    return 0;
}
```

listaCadastro[0]	char nome[50]
	int idade
	char telefone[16]
listaCadastro[1]	char nome[50]
	int idade
	char telefone[16]
...	
listaCadastro[99]	char nome[50]
	int idade
	char telefone[16]

ACESSANDO AS VARIÁVEIS

- ❑ Para acessar as variáveis utilizaremos o operador **ponto** (**.**) depois dos colchetes do índice do **array**.

```
int main (int argc, char *argv[]){  
    struct cadastro listaCadastro[100];  
  
    strcpy(listaCadastro[0].nome, "Fulano de Tal");  
  
    scanf("%i", &listaCadastro[0].idade); // ou listaCadastro[0].idade = 26;  
  
    strcpy(listaCadastro[0].telefone, "(011)98765-4321");  
  
    return 0;  
}
```

ATRIBUIÇÃO ENTRE STRUCTS

- ❑ Podemos copiar o conteúdo de um struct para outro diretamente (desde que sejam do “mesmo tipo”).

```
int main (int argc, char *argv[]){  
  
    struct cadastro c1, c2;  
    struct cadastro listaCadastro[100];  
  
    c2 = c1;  
    listaCadastro[1] = listaCadastro[0];  
  
    return 0;  
}
```

STRUCT DE STRUCT

- ❑ Sendo um struct um tipo de dado, podemos declarar um struct que utilize outro struct previamente definido.

```
struct endereco{  
    char rua[50];  
    int numero;  
};  
  
struct cadastro{  
    char nome[50];  
    int idade;  
    char telefone[16];  
    struct endereco end;  
};
```

ACESSANDO AS VARIÁVEIS

- ❑ Para acessar as variáveis utilizaremos novamente o operador **ponto** (**.**).

```
int main (int argc, char *argv[]){  
  
    struct cadastro c1;  
  
    strcpy(c1.nome, "Fulano de Tal");  
    scanf("%i", &c1.idade); // ou c1.idade = 26;  
    strcpy(c1.telefone, "(011)98765-4321");  
  
    strcpy(c1.end.rua, "Logo Ali");  
    c1.end.numero = 10;  
  
    return 0;  
}
```


INICIALIZAÇÃO DE STRUCTS

- ❑ Inicializando um struct de struct.

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

struct retangulo{
    struct ponto p1, p2, p3, p4;
};

int main (int argc, char *argv[]){

    struct retangulo rect = { {0,0} , {0,1} , {1,1} , {1,0} };

    return 0;
}
```

COMANDO TYPEDEF

- ❑ A linguagem C permite que o programador defina um novo nome para tipos de dados existentes, como se fosse um “apelido”. Para isso, utiliza-se o comando ***typedef***.

```
typedef tipo_existente novo_nome;
```

- ❑ O ***typedef*** é muito utilizado para definir nomes mais simples para estruturas, evitando utilizar a palavra ***struct*** sempre que referenciarmos o struct.

COMANDO TYPEDEF

❑ Exemplo:

```
struct cadastro{
    char nome[50];
    int idade;
    char telefone[16];
    struct endereco end;
};

typedef struct cadastro cad;

int main (int argc, char *argv[]){
    cad c1;
    return 0;
}
```

PARÂMETROS E RETORNO

- ❑ Um ***struct*** pode ser enviado como parâmetro passados por valor, além de definirem o tipo de retorno para funções.

```
struct complexo{  
    float a,b;  
    char forma;  
};  
  
typedef struct complexo complexo;  
  
complexo lerComplexo();  
complexo somarComplexos( complexo c1, complexo c2 );  
void exhibirComplexo (complexo c1);
```

PARÂMETROS E RETORNO

```
int main (int argc, char *argv[]){  
    complexo c1;  
  
    c1 = lerComplexo();  
    exhibirComplexo(c1);  
  
    return 0;  
}
```

PARÂMETROS E RETORNO

```
complexo lerComplexo(){
    complexo aux;
    printf("Forma (c)artesiana ou (p)olar: ");
    setbuf(stdin,0);
    scanf("%c", &aux.forma);

    if(aux.forma == 'c'){
        printf("Digite as componentes RE e IM: ");
    }
    else{
        printf("Digite o modulo e a fase: ");
    }

    scanf("%f,%f", &aux.a, &aux.b);
    return aux;
}
```

PARÂMETROS E RETORNO

```
void exibirComplexo (complexo c){  
    if(c.forma == 'c'){  
        printf("%.4f %c %.4f j\n", c.a, c.b>0?'+':'-',fabs(c.b));  
    }  
    else{  
        printf("%.4f < %.4f\n", c.a, c.b );  
    }  
}
```

ATIVIDADE

- ❑ Abra a pasta Downloads no VS Code
 - ❑ Utilize o comando `git clone https://github.com/ECM404/lista6.git --recursive`
 - ❑ Entre na pasta utilizando o comando `cd lista6`
 - ❑ Inicialize o diretório com o comando `make install`