



REGISTOS

struct


Um **vector** agrupa dados do mesmo tipo.
Um **registo** agrupa dados de tipos diferentes.

Registos ajudam a organizar dados complexos pois permitem tratar um conjunto de variáveis relacionadas como uma unidade.

abstração

nCal, nome, ano, notas são etiquetas


```
struct Aluno {  
    int nCAL;  
    char nome[40];  
    int ano;  
    float notas[3];  
};
```



declarar
registo

nCal, nome, ano, notas são etiquetas

```
struct Aluno {  
    int nCAL;  
    char nome[40];  
    int ano;  
    float notas[3];  
};
```



declarar
registo

//dentro da main

```
struct Aluno a = {1743, "Asimov", 6, {20.0, 20.0, 20.0}};
```

nCal, nome, ano, notas são etiquetas

```
struct Aluno {  
    int nCAL;  
    char nome[40];  
    int ano;  
    float notas[3];  
};
```

} declararam registro

//dentro da main

```
struct Aluno a = {1743, "Asimov", 6, {20.0, 20.0, 20.0}};
```

1743		'A'	's'	'i'	'm'	'o'	'v'	'\0'	
1000	...	1004	1005	1006	1007	1008	1009	1010	...

	6		20.0		20.0		20.0		
...	1044	...	1048	...	1052		1056	...	1060

nCal, nome, ano, notas são etiquetas

```
struct Aluno {  
    int nCAL;  
    char nome[40];  
    int ano;  
    float notas[3];  
};
```



a.nCAL	1743
a.nome	1004
a.nome[1]	's'
a.notas	1048
p	1000
p->ano	6

//dentro da main

```
struct Aluno a = {1743, "Asimov", 6, {20.0, 20.0, 20.0}};  
struct Aluno *p = &a;
```

1743		'A'	's'	'i'	'm'	'o'	'v'	'\0'	
1000	...	1004	1005	1006	1007	1008	1009	1010	...

	6		20.0		20.0		20.0		
...	1044	...	1048	...	1052		1056	...	1060

```

struct Aluno {
    int nCAL;
    char nome[40];
    int ano;
    float notas[3];
};

```

//dentro da main

```

struct Aluno a = {1743, "Asimov", 6, {20.0, 20.0, 20.0}};

```

```

struct Aluno *p = &a;

```

```

a.nCal = 1742;

```

```

a.notas[1] = 12.5;

```

} acesso a campo
do registro

a.nCAL	1743
a.nome	1004
a.nome[1]	's'
a.notas	1048
p	1000
p->ano	6

1742		'A'	's'	'i'	'm'	'o'	'v'	'\0'	
1000	...	1004	1005	1006	1007	1008	1009	1010	...

	6		20.0		12.5		20.0		
...	1044	...	1048	...	1052		1056	...	1060

Comparação

```
struct Aluno{
    int nCAL;
    char nome[40];
    int ano;
    float notas[3];
};
//dentro da main
struct Aluno a = {1743, "Asimov", 6, {20.0, 20.0, 20.0}};
struct Aluno b = {1642, "Newton", 6, {20.0, 20.0, 20.0}};

if (a == b) printf("alunos iguais");
```



Comparação de
registos tem de ser
feita campo a campo.

```
if (a.nCal == b.nCal &&
    strcmp(a.nome, b.nome) == 0 &&
    a.ano == b.ano ...)
    printf("alunos iguais");
```

Registos dentro de registos

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

```
struct rect{
    struct ponto inf_esq;
    struct ponto sup_dir;
};
```

```
//dentro da main
```

```
struct ponto p1 = {2, 2};
struct ponto p2 = {4, 3};
struct rect r = {p1, p2};
```

```
r.inf_esq.x = 0;
```

```
// p1 nao alterado
```

```
// pois r.inf_esq é uma
cópia de p1, e não p1 em si
```

Registos e funções

```
struct ponto novoponto(int x, int y){  
    struct ponto temp;  
    temp.x = x;  
    temp.y = y;  
    return temp;  
}
```

```
struct rect novorect(struct ponto p1, struct ponto p2){  
    struct rect temp;  
    temp.inf_esq = p1;  
    temp.sup_dir = p2;  
    return temp;  
}
```

Registos e funções

```
struct rect novorect(struct ponto p1, struct ponto p2){  
    struct rect rtemp;  
    rtemp.inf_esq = p1;  
    rtemp.sup_dir = p2;  
    return temp;  
}
```

cópias de pa e pb

```
//dentro da main  
struct ponto pa = {2, 2};  
struct ponto pb = {4, 3};  
struct rect r = novorect(pa, pb);
```

cópia de temp ←

typedef

typedef serve para dar novos nomes a tipos de dados existentes.

```
typedef <tipo existente> <novo_nome>
```

typedef

typedef serve para dar novos nomes a tipos de dados existentes.

```
typedef <tipo existente> <novo_nome>
```

O tipo existente continua a estar disponível através do seu nome original.

typedef

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



```
typedef struct{  
    int x;  
    int y;  
} Ponto;
```

```
struct rect{  
    struct ponto inf_esq;  
    struct ponto sup_dir;  
};
```



```
typedef struct{  
    Ponto inf_esq;  
    Ponto sup_dir;  
} Rect;
```

typedef

```
typedef struct{  
    int x;  
    int y;  
} Ponto;
```

```
typedef struct{  
    Ponto inf_esq;  
    Ponto sup_dir;  
} Rect;
```

```
Rect novorect(Ponto p1, Ponto * p2){  
    Rect temp;  
    temp.inf_esq = p1;  
    temp.sup_dir = *p2;  
    return temp;  
}
```

```
//dentro da main  
Ponto pa = {2, 2};  
Ponto pb = {4, 3};  
Rect r = novorect(pa, &pb);
```



```
Rect novorect(Ponto p1, Ponto * p2) {  
    Rect temp;  
    temp.inf_esq = p1;  
    temp.sup_dir = *p2;  
    return temp;  
}
```

```
//dentro da main  
Ponto pa = {2, 2};  
Ponto pb = {4, 3};  
Rect r = novorect(pa, &pb);
```

O ponto `pa` é copiado para `p1` (argumento da função), que depois é copiado para `temp.inf_esq`.

O ponto `pb` é passado por referência para a função e depois é copiado para `temp.sup_dir`.

Assim, `pa` é copiado 2x, enquanto `pb` só é copiado 1x.

Usar apontadores de registos reduz significativamente a quantidade de cópias de dados e torna o programa mais rápido.

apontadores e registos

```
typedef struct {
    char matricula[6];
    char marca[20];
    char modelo[30];
    int tipo;
} Veiculo;
```

```
typedef struct {
    int id;
    char descricao[500];
    int n_veiculos;
    Veiculo veiculos[50];
} Frota;
```

```
void mudaMatricula(Frota * f, int i, char * m){
    int j;
    for(j=0; j<6; j++)
        f->veiculos[i].matricula[j] = m[j];
```

```
//dentro da main }
```

```
Frota f1;
```

```
f1.id = 1;
```

```
Veiculo v = {"31RI59", "DeLorean", "DMC-12", 1};
```

```
f1.veiculos[0] = v;
```

```
f1.n_veiculos = 1;
```



Exercício 1

Ver enunciado nos slides novos.