

Registos (Estruturas)

Taxonomia de tipos de dados

Simplex

- Numéricos
 - int (inteiros)
 - float (reais)
 - char (carateres)
- Apontadores
 - *

Compostos

- Padrão
 - arrays (1 dimensão e 2 dimensões)
 - FILE (ficheiros)
- Definidos pelo utilizador
 - **struct** (registos/estruturas)

Tipos de dados compostos

Definição pelo utilizador

- Não pertencem ao léxico da linguagem
- Requerem a utilização de
 - um mecanismo sintático, ou
 - uma palavra reservada ([], **struct**, FILE, ...)

Composição

- Têm valores compostos
- Exemplo (estudantes de uma UC de um curso):
ESTUDANTE A = { 13543, 345, (14.50, 13.75) };
ESTUDANTE B = { 13654, 345, (12.25, 8.75) };

Não têm ordem nem escala

- Exemplo:
 - $A < B$ (não faz sentido)

Estruturas (struct)

Motivação

- Necessidade de guardar informação composta por vários tipos de dados numa só entidade (estrutura)
- Exemplo
 - necessidade das organizações de inserir, alterar, eliminar e consultar dados em formulários
 - há 3 dados comuns a todos os sistemas de informação: nome, morada e telefone
 - em linguagem C, pode-se agrupar aqueles dados num novo tipo de dados (CLIENTE)

```
typedef struct {  
    char Nome[60];  
    char Morada[100];  
    int Telefone;  
} CLIENTE;
```

Definição

- É uma coleção ou conjunto de elementos (ou variáveis) armazenados numa zona contígua de memória
- Os elementos são variáveis usuais identificadas pelo nome
- Os elementos são também designados por **campos**, **membros** ou **componentes**
- Ao contrário dos arrays
 - os elementos não são necessariamente do mesmo tipo de dados
 - os elementos não são indexados
- Exemplo:

```
typedef struct {  
    int Numero;  
    int CodigoUC;  
    float Notas[2];  
} ESTUDANTE;
```



Sintaxe (sem typedef)

```
struct nome {  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipoN campoN;  
};
```

em que

- **struct**: palavra reservada associado ao tipo de dados composto registo/estrutura
- **nome**: identificador do **tipo estrutura**
- **campoK** (K = 1, ..., N): identificador do nome do campo K da estrutura
- **tipoK** (K = 1, ..., N): tipo de dados do campo K da estrutura

Sintaxe (com typedef)

```
typedef struct {  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipoN campoN;  
} nome;
```

em que

- **typedef**: palavra reservada associado a tipo definido pelo utilizador
- **struct**: palavra reservada associado ao tipo de dados composto registo/estrutura
- **nome**: identificador do **tipo de dados estrutura** (tipo definido pelo utilizador)
- **campoK** ($K = 1, \dots, N$): identificador do nome do campo K da estrutura
- **tipoK** ($K = 1, \dots, N$): tipo de dados do campo K da estrutura

Declaração de variáveis

- Definição de tipo estrutura e declaração de variáveis, no mesmo momento
- Exemplo

```
struct ESTUDANTE {  
    int Numero;  
    int CodigoUC;  
    float Notas[2];  
} A;
```

- declaração da variável A do tipo ESTUDANTE

Declaração de variáveis

- Definição de tipo estrutura e declaração de variáveis, em momentos diferentes
- Exemplo

```
struct ESTUDANTE {  
    int Numero;  
    int CodigoUC;  
    float Notas[2];  
};  
struct ESTUDANTE A;
```

- definição do tipo estrutura ESTUDANTE
- declaração da variável A do tipo ESTUDANTE

Declaração de variáveis

- Definição de tipo estrutura sem nome e declaração de variáveis, no mesmo momento
- Exemplo

```
struct {  
    int Numero;  
    int CodigoUC;  
    float Notas[2];  
} A;
```

- declaração da variável A como uma estrutura

Declaração de variáveis

- Definição de tipo de dados estrutura com **typedef** e declaração de variáveis, em momentos diferentes
- Exemplo

```
typedef struct {  
    int Numero;  
    int CodigoUC;  
    float Notas[2];  
} ESTUDANTE;  
void main()  
{  
    ESTUDANTE A;  
    ...  
}
```

Caraterísticas específicas

- Contiguidade
 - ocupa uma zona *contígua* de memória
- Não são homogeneamente tipadas
 - as componentes podem ter diferentes tipos de dados
- Acesso por identificador
 - cada componente tem um identificador
- Exemplo (ESTUDANTE)

```
typedef struct {  
    int Numero;  
    int CodigoUC;  
    float Notas[2];  
} ESTUDANTE;
```

Operadores de acesso

- Operador "."

```
ESTUDANTE A;  
A.Numero = 13543;  
A.CodigoUC = 345;  
A.Notas[0] = 14.50;  
A.Notas[1] = 13.75;
```

Operadores de acesso

- Operador "->"

```
ESTUDANTE *A;  
A = (ESTUDANTE *) malloc(sizeof(ESTUDANTE));  
A->Numero = 13543;  
A->CodigoUC = 345;  
A->Notas[0] = 14.50;  
A->Notas[1] = 13.75;
```

que é equivalente a

```
ESTUDANTE *A;  
A = (ESTUDANTE *) malloc(sizeof(ESTUDANTE));  
(*A).Numero = 13543;  
(*A).CodigoUC = 345;  
(*A).Notas[0] = 14.50;  
(*A).Notas[1] = 13.75;
```

Operador de atribuição

- Considere-se a seguinte declaração

```
ESTUDANTE A, B;
```

- A atribuição pode ser feita campo a campo

```
A.Numero = 13543;  
A.CodigoUC = 345;  
A.Notas[0] = 14.50;  
A.Notas[1] = 13.75;
```

- A atribuição também pode ser feita entre estruturas, **como** para qualquer tipo de dados simples (e **ao contrário** dos arrays)

```
B = A;
```

Exemplo

```
#include <stdio.h>
typedef struct {
    int Numero;
    int CodigoUC;
    float Notas[2];
} ESTUDANTE;
```

```
void main() {
    ESTUDANTE A, B;
    A.Numero = 13543;
    A.CodigoUC = 345;
    A.Notas[0] = 14.50;
    A.Notas[1] = 13.75;
    B = A;
    printf("%d\n%d\n%f e %f\n", B.Numero, B.CodigoUC, B.Notas[0], B.Notas[1]);
}
```

Saída (monitor):

```
13543
345
14.50 e 13.75
```

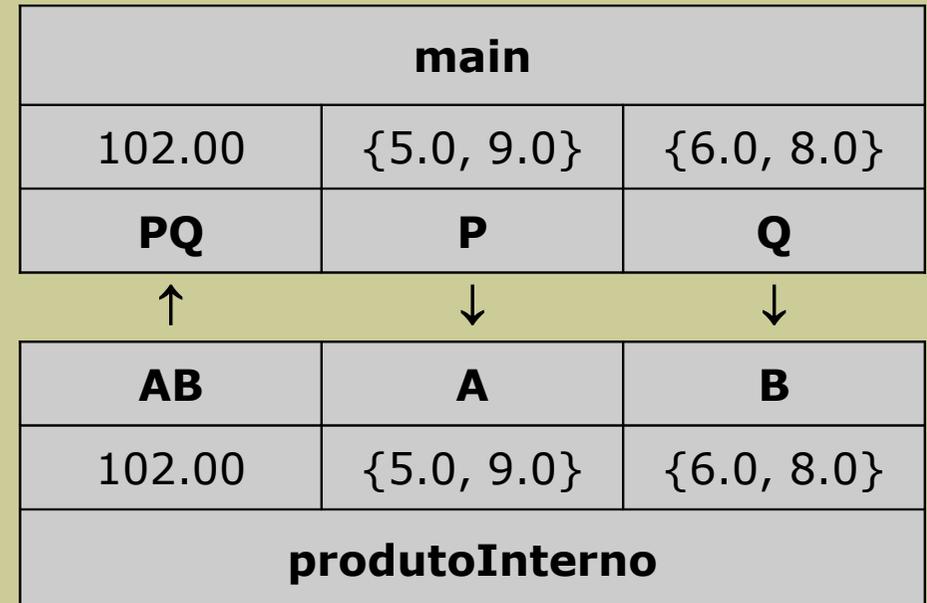
Passagem de estruturas para funções

- Passagem por cópia de valor da estrutura
 - o operador que deve ser usado no corpo da função para aceder aos membros da estrutura é o `.` (`nome.campo`)
- Passagem por cópia de valor de endereço da estrutura
 - o operador que deve ser usado no corpo da função para aceder aos membros da estrutura é o `->` (`nome->campo`)

Passagem de estruturas para funções

- Passagem por cópia de valor da estrutura (exemplo)

```
#include <stdio.h>
typedef struct {
    float X;
    float Y;
} PONTO;
float produtoInterno (PONTO A, PONTO B) {
    float AB;
    AB = A.X * B.X + A.Y * B.Y;
    return AB;
}
void main() {
    PONTO P = { 5.0, 9.0 }, Q = { 6.0, 8.0 };
    float PQ = produtoInterno(P, Q);
    printf("Produto Interno = %f\n", PQ);
}
```



Passagem de estruturas para funções

- Passagem por cópia de valor de endereço da estrutura (exemplo)

```
#include <stdio.h>
typedef struct {
    float X;
    float Y;
} PONTO;
float produtoInterno (PONTO *A, PONTO *B) {
    float AB = A->X * B->X + A->Y * B->Y;
    // float AB = (*A).X * (*B).X + (*A).Y * (*B).Y;
    return AB;
}
void main() {
    PONTO P = { 5.0, 9.0 }, Q = { 6.0, 8.0 };
    float PQ = produtoInterno(&P, &Q);
    printf("Produto Interno = %f\n", PQ);
}
```

	...	
200500	{5.0, 9.0}	P
200508	{6.0, 8.0}	Q
200516	102.00	PQ
	...	

main		
102.00	200500	200508
PQ	&P	&Q
↑	↓	↓
AB	A	B
102.00	200500	200508
produtoInterno		

Passagem de estruturas para subprogramas

Passagem de estruturas

- passagem por valor:
 - passa-se o nome da estrutura, o que implica a cópia do conteúdo da estrutura para uma estrutura local ao subprograma
- passagem por referência:
 - passa-se o endereço da estrutura, o que implica a cópia do endereço da estrutura para uma variável apontadora local ao subprograma

Passagem de arrays (relembrar)

- passagem por valor: não existe
 - não é possível copiar ou passar o conteúdo de um array para um array local ao subprograma (ao passar o nome do array, passa o endereço do seu primeiro elemento)
- passagem por referência:
 - passa-se o endereço do array (que é do seu primeiro elemento), o que implica a cópia do endereço do array para uma variável apontadora local ao subprograma

Exemplo - Estruturas

- Passagem por valor
 - recebe cópias de duas estruturas:

```
float produtoInterno (PONTO A, PONTO B);
```

- Passagem por referência
 - recebe dois endereços de estruturas:

```
float produtoInterno (PONTO *A, PONTO *B);
```

Exemplo - Arrays

- Passagem por referência
 - recebe os valores dos endereços de dois arrays:

```
float produtoInterno (float A[], float B[]);  
float produtoInterno (float *A, float *B);
```

Exemplo - Arrays

- Passagem por referência
 - passagem por cópia do valor de endereço de array

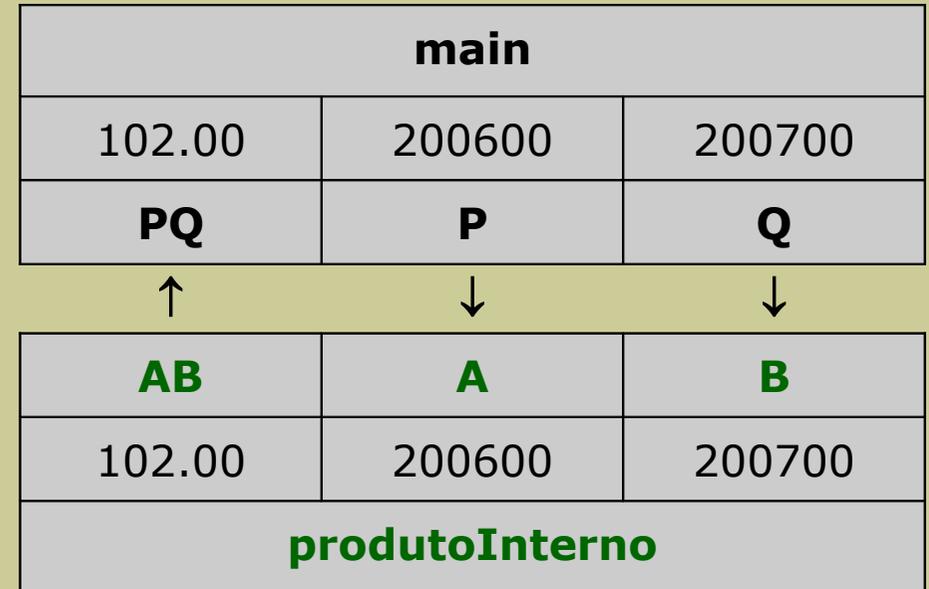
```
#include <stdio.h>

float produtoInterno (float A[], float B[]) { // ou float *A, float *B
    float AB;
    AB = A[0].X * B[0].X + A[1].Y * B[1].Y;
    return AB;
}

void main() {
    float P[2] = { 5.0, 9.0 }, Q[2] = { 6.0, 8.0 };
    float PQ;
    PQ = produtoInterno(P, Q);
    printf("Produto Interno = %f\n", PQ);
}
```

Exemplo - Arrays

...		
200400	200600	P
...		
200500	200700	Q
...		
200600	5.0	P[0] ⇔ A[0]
200604	9.0	P[1] ⇔ A[1]
...		
200700	6.0	Q[0] ⇔ B[0]
200704	8.0	Q[1] ⇔ B[1]
...		
200800	102.00	PQ
...		



Retorno de uma estrutura por um subprograma

- Um subprograma pode devolver uma estrutura de duas formas:
 - passagem **por valor**
 - passagem **por referência**
- **Passagem por valor**
 - devolve-se o nome da estrutura, o que implica a cópia da estrutura local ao subprograma para uma estrutura local ao subprograma que a chama (ex: main)
- **Passagem por referência**
 - devolve-se o endereço da estrutura local ao subprograma para uma variável apontadora que é local ao subprograma que a chama (ex: main)

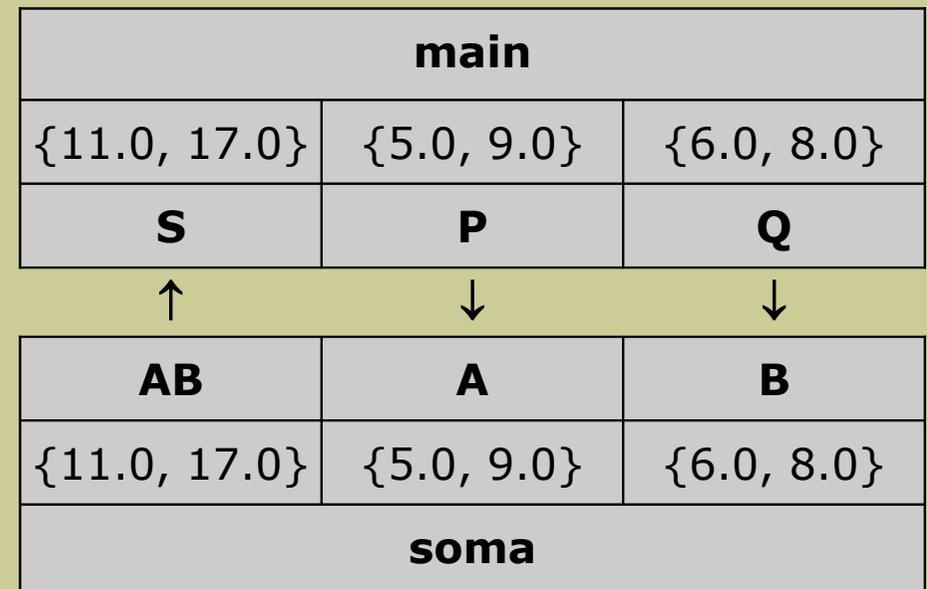
Retorno de uma estrutura por um subprograma

- Exemplo de retorno por valor

```
#include <stdio.h>
typedef struct {
    float X;
    float Y;
} PONTO;
PONTO soma (PONTO A, PONTO B) {
    PONTO AB;
    AB.X = A.X + B.X;
    AB.Y = A.Y + B.Y;
    return AB;
}

void main() {
```

```
    PONTO P = { 5.0, 9.0 }, Q = { 6.0, 8.0 };
    PONTO S = soma(P, Q);
    printf("Vetor soma = (%f %f)", S.X, S.Y);
}
```

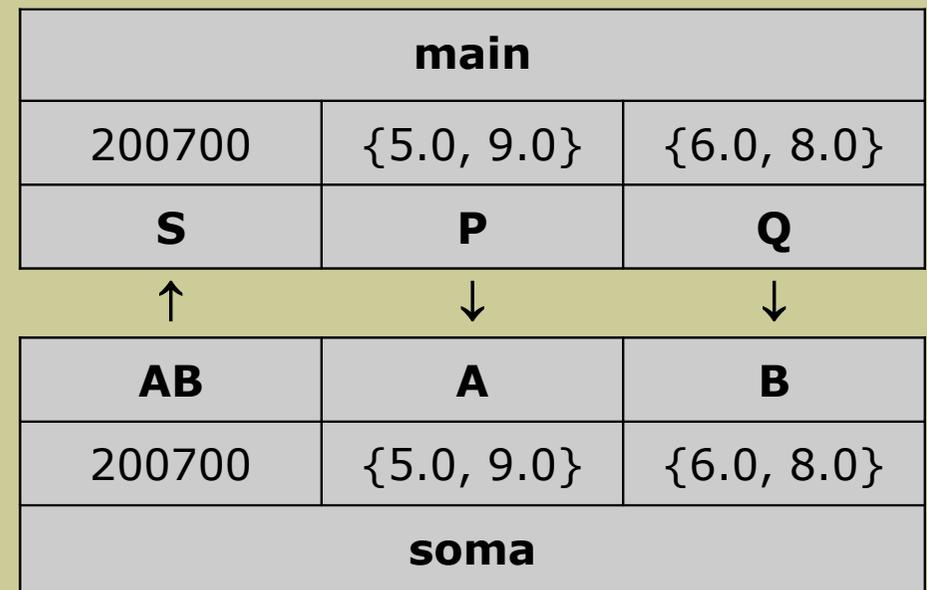


Retorno de uma estrutura por uma função

- Exemplo de retorno por referência

```
#include <stdio.h>
typedef struct {
    float x;
    float y;
} PONTO;
PONTO *soma (PONTO A, PONTO B)
{
    PONTO *AB;
    AB = malloc(sizeof(PONTO));
    AB->x = A.X + B.X;
    AB->y = A.Y + B.Y;
    return AB;
}
```

```
void main() {
    PONTO P = { 5.0, 9.0 }, Q = { 6.0, 8.0 };
    PONTO *S = soma(P, Q);
    printf("Soma = (%f %f)", S->X, S->Y);
}
```



Array de estruturas

Exemplo

```
#include <stdio.h>
typedef struct {
    int Numero;
    int CodigoUC;
    float Notas[2];
} ESTUDANTE;
```

```
void main()
{
    ESTUDANTE PROG[100];
    int k;
    for (k = 0; k < 100; k++) {
        printf("Insira o numero: ");
        scanf("%d", &PROG[k].Numero);
        printf("Insira o codigo da UC: ");
        scanf("%d", &PROG[k].CodigoUC);
        printf("Insira as notas: ");
        scanf("%f%f", &PROG[k].Notas[0], &PROG[k].Notas[1]);
    }
}
```