Algoritmia e

Programação

Enumerações, estruturas e uniões.



Enumerações

•

•

Uma enumeração é um tipo definido pelo programador, que permite atribuir nomes a constantes inteiras (i.e. tipo **int**).

As enumerações permitem escrever programas que são mais fáceis de ler e de manter, utilizando ***nomes*** em vez de ***números mágicos***.

* Uma enumeração é declarada com a palavra chave **enum**.



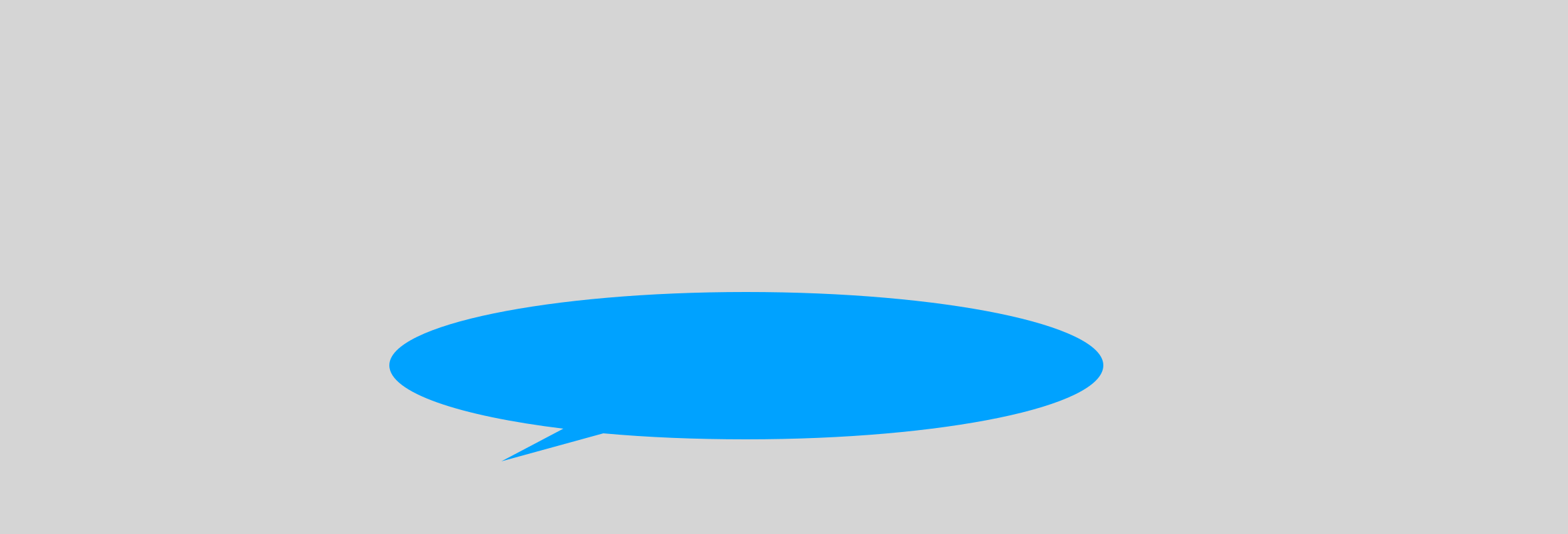
**APROG - LETI**

Declarar uma enumeração

* O compilador atribui **automaticamente** valores **sequenciais** aos nomes declarados.
* O primeiro valor, por omissão, é o ZERO. **enum** estado

{

DESLIGADO, /\* DESLIGADO = 0 \*/



LIGADO /\* LIGADO = 1 \*/

}; **maquina** é uma variável do

tipo **enum estado**

**enum** estado maquina;



**APROG - LETI**

Declarar uma enumeração

* O programador pode atribuir valores específicos aos identificadores.



**enum** estado

{

|  |  |  |  |
| --- | --- | --- | --- |
| OK = 10, | /\* | OK=10 | \*/ |
| ERRO\_SENSOR = 20, | /\* | ERRO\_SENSOR = 20 | \*/ |
| ERRO\_RADIO, | /\* | ERRO\_RADIO = 21 | \*/ |
| TEMPERATURA\_OK = 50, /\* | | TEMPERATURA\_OK = 50 \*/ | |

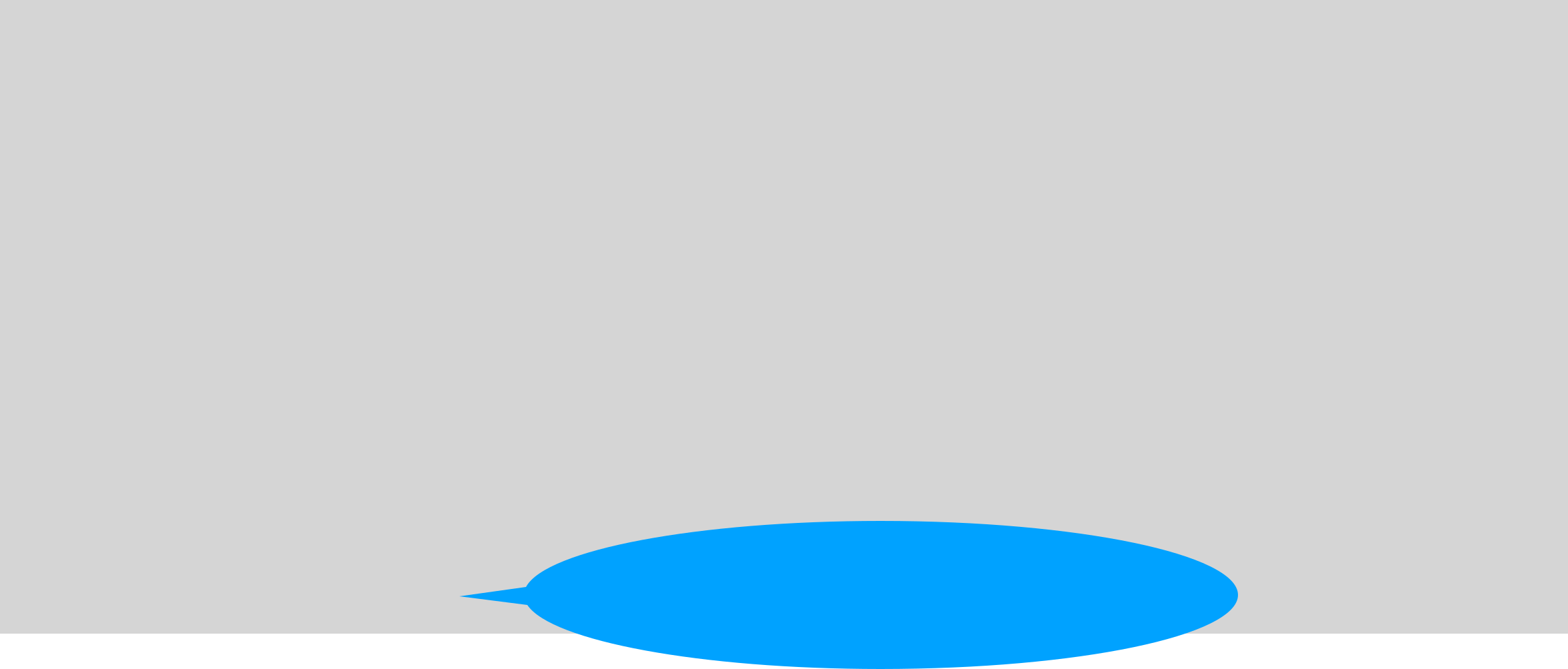
};



**APROG - LETI**

Declarar um tipo baseado numa enumeração

* Por vezes é útil criar um tipo baseado numa enumeração.



|  |  |  |  |
| --- | --- | --- | --- |
| **typedef enum** |  |  |  |
| { |  |  |  |
| OK = 10, | /\* | OK=10 | \*/ |
| ERRO\_SENSOR = 20, | /\* | ERRO\_SENSOR = 20 | \*/ |
| ERRO\_RADIO, | /\* | ERRO\_RADIO = 21 | \*/ |
| TEMPERATURA\_OK = 50, /\* | | TEMPERATURA\_OK = 50 \*/ | |
| } estado\_t; |  |  |  |

|  |  |  |
| --- | --- | --- |
| estado\_t maquina; | **maquina** é uma variável do |  |
| tipo **estado\_t** |  |
|  |  |



**APROG - LETI**

Utilização de enumerações



estado\_t maquina; // Criada a variável máquina.

maquina = ERRO\_SENSOR; // Atribuição de valor à variável. /\* Mais algumas instruções... \*/ if(maquina != OK) {

/\* Foi detectado um erro na máquina... \*/

}

/\* E o programa continuaria... \*/



**APROG - LETI**

Estruturas

* Uma estrutura é um tipo definido pelo programador, que permite definir um *registo* que consiste numa sequência de membros.

•

•

Cada membro é identificado por um *nome*.

Cada membro tem um tipo que é independente dos outros membros.

* As estruturas permitem agrupar valores que são logicamente coesos num único tipo.
  + Por exemplo, as coordenadas de um ponto são logicamente coesas e podem ser agrupadas numa estrutura.
* Uma estrutura é declarada com a palavra chave **struct**.



**APROG - LETI**

Declarar uma estrutura



**struct** ponto

{

float x;

float y;

float z;

};

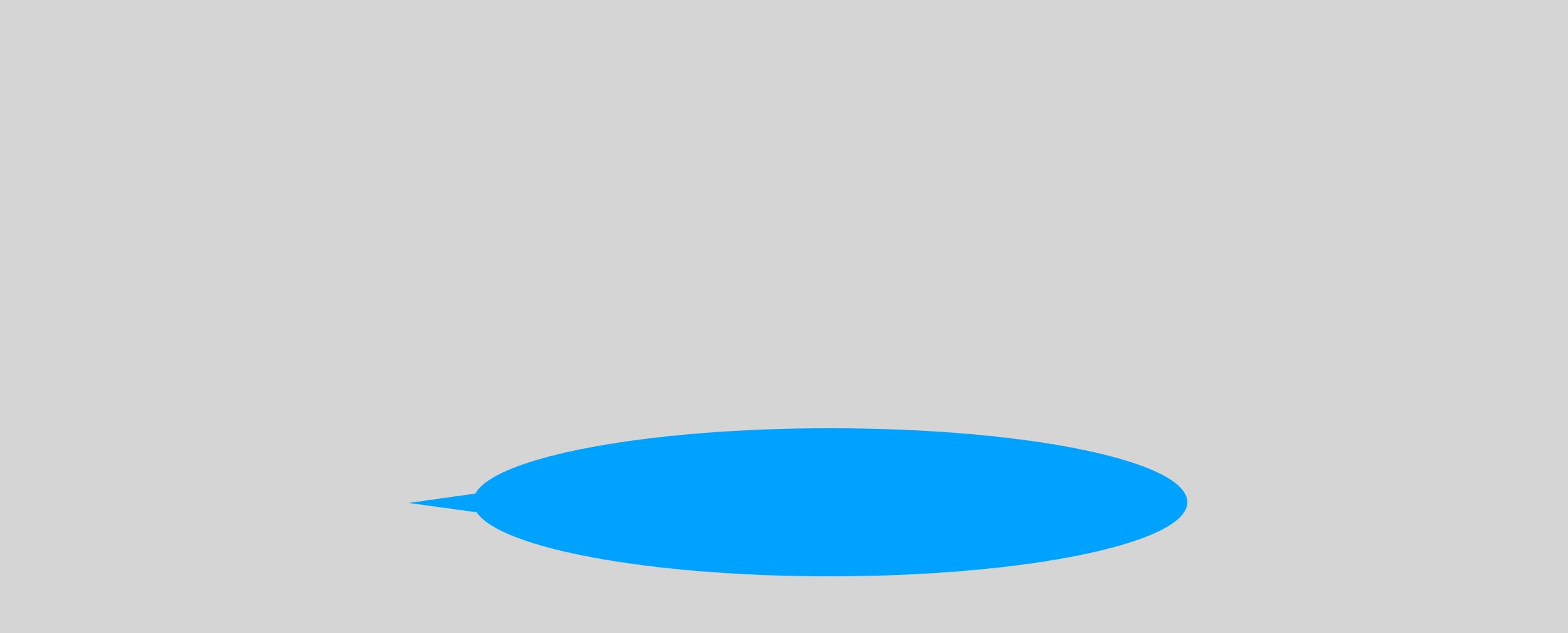
**struct** ponto p1, p2; // Variáveis p1 e p2 guardam pontos.



**APROG - LETI**

Declarar um tipo baseado numa estrutura

* Por vezes é útil criar um tipo baseado numa estrutura.



**typedef struct**

{

float x;

float y;

float z;

} ponto\_t; ponto\_t p1, p2;

**p1 e p2** são variáveis do tipo

**ponto\_t**



**APROG - LETI**

Aceder aos membros de uma estrutura

* Os membros são acedidos através do seu identificador, precedido pelo operador **.** (ponto).



ponto\_t p1, p2;

p1.x = 1.0;

p1.y = 2.0;

p1.z = 10.6;

p2 = p1;

p2.z = p2.z - 2.5 ;



**APROG - LETI**

Uniões

* Uma união é um tipo definido pelo programador, em que permite definir membros que se **sobrepõem**.
  + Cada membro é identificado por um *nome*.
  + Cada membro tem um tipo que é independente dos outros membros.
* As uniões só permitem ter um membro, em cada instante.
  + Útil quando queremos um tipo que possa variar.
  + O programador tem que manter o controlo de qual o membro que está deve ser referenciado a cada instante.
* Uma estrutura é declarada com a palavra chave **union**.



**APROG - LETI**

Declarar uma união



**union** uniao

{

int x;

float y;

};

**struct** uniao u;

u.x = 10; /\* Pode guardar um inteiro... \*/ u.y = 5.5; /\* ... ou pode guardar um real. \*/



**APROG - LETI**

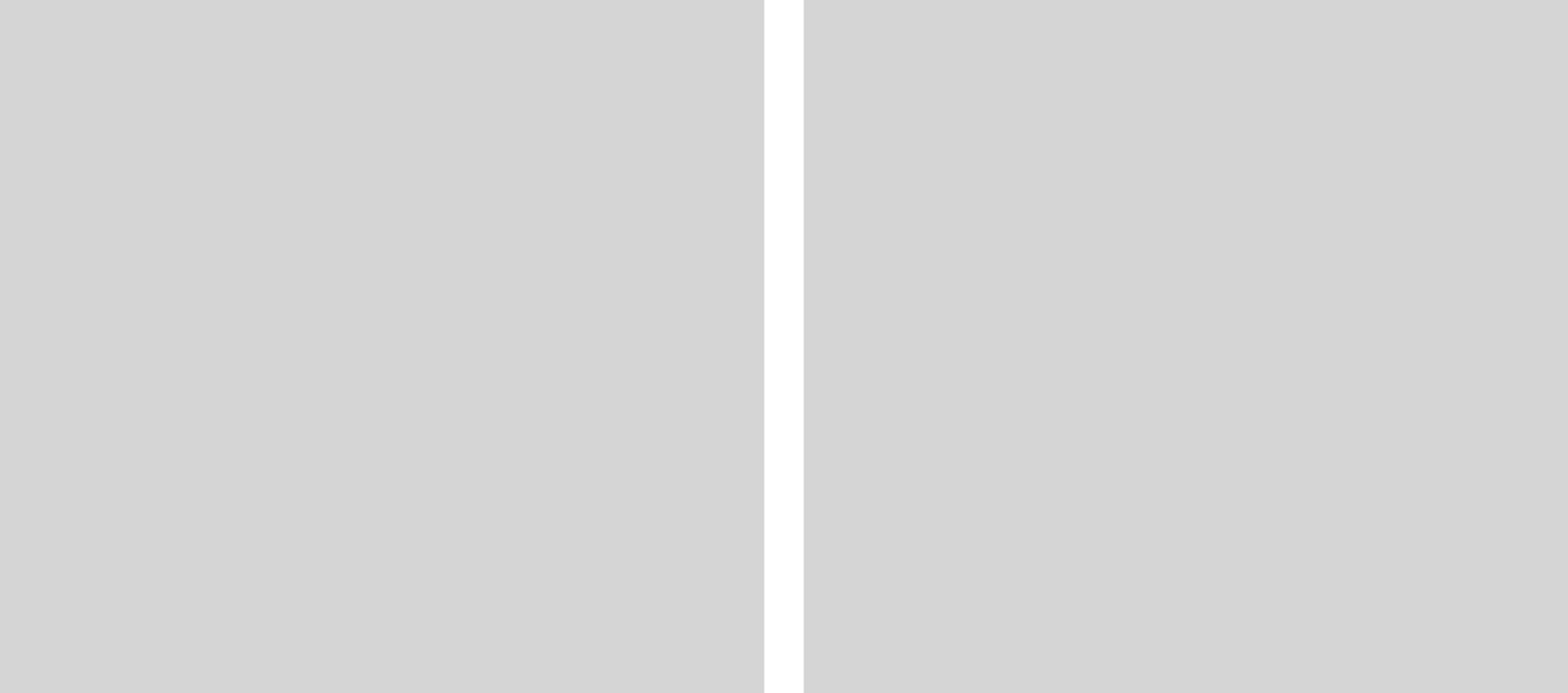
Como saber que tipo de valor está numa união?

* A linguagem C é muito permissiva, sendo possível:
  1. guardar um valor num membro de uma união
  2. e ler de outro membro da mesma união.
* Esta situação resulta em erros de má interpretação dos bits guardados na união.
* O programador **tem que** controlar o **membro activo** da união!



**APROG - LETI**

Exemplo de má interpretação do valor



**union** exemplo {

char c;

int x;

};

**union** exemplo teste;

teste.x = 1000;

printf("int: %d\n", teste.x);

printf("char: %hhd\n", teste.c);

printf("int (hex): %x\n", teste.x);

printf("char (hex): %hhd\n", teste.c);

***< -- RUN clang 5.0 (C11) -->***

int: 1000

char: -24

int (hex): 3e8

char (hex): e8



**APROG - LETI**

Controlo de uma união



|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Este membro permite |  |
| **typedef enum** | **typedef struct** | identificar o tipo de |  |
| leitura na mensagem. |  |
| { | { |  |  |
| MSG\_TEMPERATURA, | msg\_tipo\_t tipo; | |  |
| MSG\_PRESSAO, | **union** |  |  |
| MSG\_HUMIDADE | { |  |  |
| } msg\_tipo\_t; | **float** temperatura; | |  |
|  | **int** pressao; | |  |



**int** humidade;

} leitura;

} mensagem\_sensor\_t;

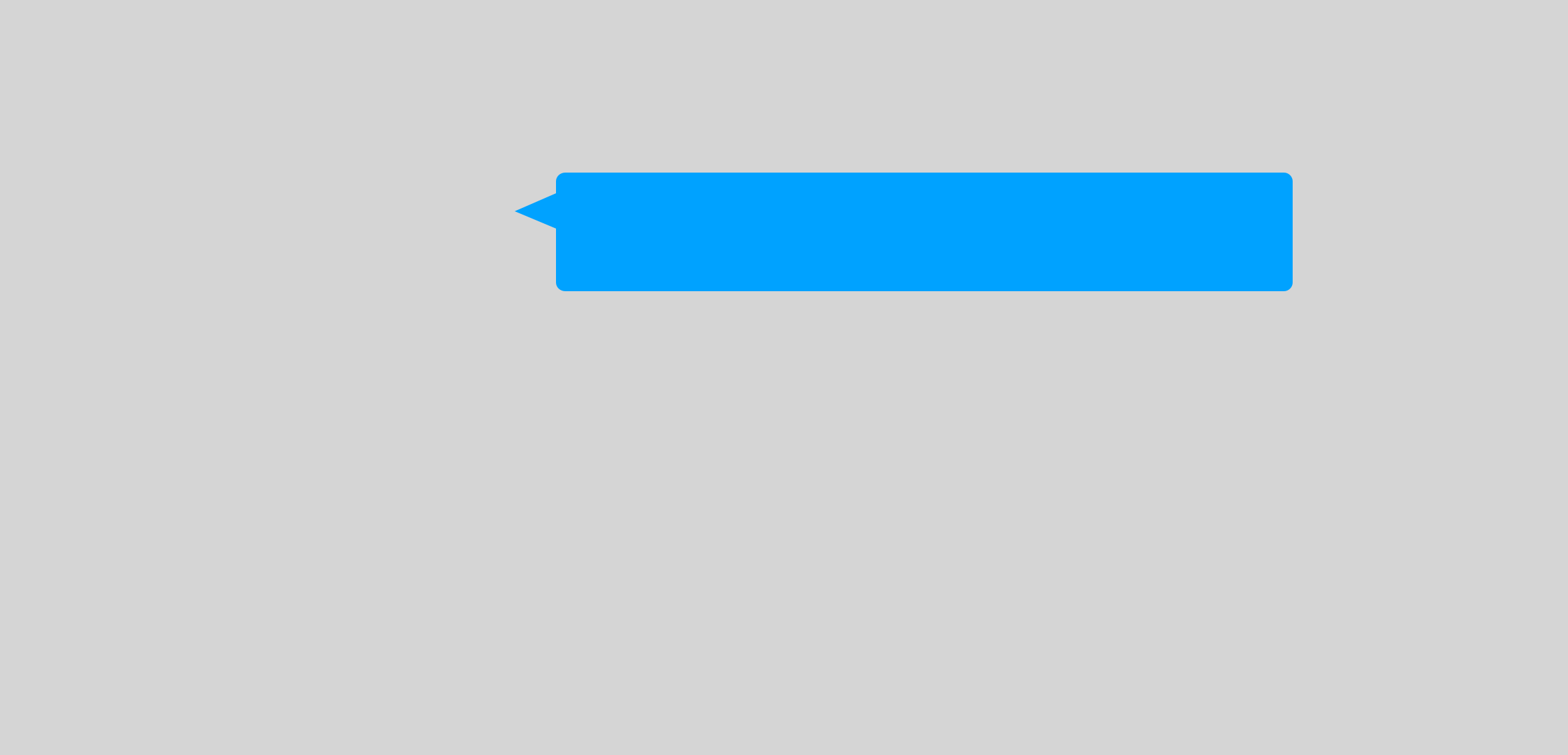


**APROG - LETI**

O valor neste membro permite determinar

qual o membro activo da união.

Controlo de uma união



mensagem\_sensor\_t msg;

/\* Após leitura de uma medida ... \*/

**switch**(msg.tipo) { **case** MSG\_TEMPERATURA:

printf("Temperatura: %.1f\n", msg.leitura.temperatura);

**break**;

**case** MSG\_PRESSAO:

printf("Pressão: %d\n", msg.leitura.pressao); **break**;

/\* Tratar outros tipos... \*/

}



**APROG - LETI**