TIPI ENUMERATIVI

Un ***tipo enumerativo*** è un tipo di dato che consiste in un insieme ristretto di valori.

Ogni elemento appartenente al tipo ha un proprio nome (identificatore).

I nomi associati agli elementi del tipo sono trattati come costanti, chiamate costanti enumerative.

I tipi enumerativi sono compatibili con un dato intero.

La loro funzione è semplicemente quella di rendere più leggibile il codice.

TIPI ENUMERATIVI

Un ***tipo enumerativo*** viene specificato tramite ***l'elenco dei valori*** che i dati di quel tipo possono assumere

Schema generale:

```
typedef enum {
     a1, a2, a3, ... , aN } EnumType;
```

Il compilatore associa a ciascun "identificativo di valore" a_1, .., a_N un *numero naturale* (0,1,...), che viene usato nella valutazione di espressioni che coinvolgono il nuovo tipo

TIPI ENUMERATIVI

Gli "identificativi di valore" $a_1, \ldots, a_N$
sono a tutti gli effetti delle ***nuove costanti***

Esempi:

```
typedef enum {
  lu, ma, me, gi, ve, sa, dom} Giorni;
typedef enum {
  cuori, picche, quadri, fiori} Carte;
Carte   C1, C2, C3, C4, C5;
Giorni  Giorno;
if (Giorno == dom) /* giorno festivo */
else /* giorno feriale */
```

TIPI ENUMERATIVI

Analogamente si può utilizzare un tag:

```
enum Giorni{
  lu, ma, me, gi, ve, sa, dom};
enum Carte {
  cuori, picche, quadri, fiori};
enum Carte      C1, C2, C3, C4, C5;
enum Giorni     Giorno;
```

Oppure direttamente la definizione delle variabili:

```
enum {
  lu, ma, me, gi, ve, sa, dom } Giorno;
enum {
  cuori, picche, quadri, fiori} C1, C2, C3, C4, C5;
```

TIPI ENUMERATIVI

Un "identificativo di valore" può comparire *una sola volta* nella definizione di *un solo tipo*, nello stesso scope di visibilità, altrimenti si ha ambiguità

Esempio:

```
typedef   enum {
  lu, ma, me, gi, ve, sa, dom} Giorni;

typedef enum { lu, ma, me} PrimiGiorni;
```

La definizione del secondo tipo enumerativo è ***scorretta***, perché gli identificatori `lu, ma, me` sono già stati usati altrove.

TIPI ENUMERATIVI

Un tipo enumerativo è *totalmente ordinato:*
vale l'ordine con cui gli identificativi di valore sono stati elencati nella definizione

Esempio:

```
typedef enum {
  lu, ma, me, gi, ve, sa, dom} Giorni;
```

Data questa definizione,

`lu < ma` *è vera*

`lu >= sa` *è falsa*

in quanto `lu` ↔ 0, `ma` ↔ 1, `me` ↔ 2, ...

TIPI ENUMERATIVI

Poiché un tipo enumerativo è, *per la macchina C,* indistinguibile da un intero, è possibile, anche se sconsigliato, ***mescolare interi e tipi enumerativi***

Esempio:

```
typedef enum {
   lu, ma, me, gi, ve, sa, dom} Giorni;
Giorni g;
g = 5;          /* equivale a g = sa */
```

TIPI ENUMERATIVI

È possibile ***specificare esplicitamente i valori naturali (espressioni costante intere) cui associare i simboli*** **`a1,...,aN`**

- qui, `lu` ↔ 0, `ma` ↔ 1, `me` ↔ 2, ...

```
typedef   enum {
  lu, ma, me, gi, ve, sa, dom} Giorni;
```

- qui, invece, `lu` ↔ 1, `ma` ↔ 2, `me` ↔ 3, ...

```
typedef   enum {
  lu=1, ma, me, gi, ve, sa, dom} Giorni;
```

- qui, infine, l'associazione è data caso per caso

```
typedef   enum { lu=1, ma, me=7, gi, ve,
  sa, dom} Giorni;
```

E' possibile avere costanti enumerative con lo stesso valore. Ogni assegnamento esplicito viene utilizzato come punto di partenza per gli assegnamenti successivi.

IL TIPO BOOLEAN

Il boolean non esiste in C, ma si può facilmente definire in termini di tipo enumerativo:

```
typedef enum { false, true }
Boolean;
```

Di conseguenza:

`false` $\leftrightarrow$ 0, **`true`** $\leftrightarrow$ 1

`false` $<$ **`true`**

EQUIVALENZA

- La possibilità di introdurre nuovi tipi pone il problema di ***stabilire se e quanto due tipi siano compatibili fra loro***
- **Due possibili scelte:**
  - ***equivalenza strutturale***

    tipi equivalenti se ***strutturalmente identici***
  - ***equivalenza nominale***

    tipi equivalenti se ***definiti nella stessa definizione*** oppure se ***il nome dell'uno è definito espressamente come identico all'altro***

Scelta dal C

EQUIVALENZA STRUTTURALE

Esempio di **equivalenza *strutturale***

```
typedef int  MioIntero;
typedef int  NuovoIntero;
MioIntero    A;
NuovoIntero  B;
```

I due tipi `MioIntero` e `NuovoIntero` sono equivalenti perché *strutturalmente identici* (entrambi `int` per la macchina C)

Quindi, A=B è un assegnamento lecito

EQUIVALENZA NOMINALE

- Non è il caso del C, ma è il caso, per esempio, del **Pascal**
- **Esempio di equivalenza *nominale***

```
type MioIntero = integer;
type NuovoIntero = integer;
var A: MioIntero;
var B: NuovoIntero;
```

- **I due tipi `MioIntero` e `NuovoIntero` *non sono equivalenti* perché *definiti in una diversa definizione (`A:=B` non è consentito)***

Esercizio: Sistema lineare

- Scrivere una procedura/funzione che risolva un sistema lineare di due equazioni in due incognite (metodo di Cramer)

$$\begin{cases} a1x + b1y = c1 \\ a2x + b2y = c2 \end{cases}$$

- Soluzione:

$$x = (c1b2 - c2b1) / (a1b2 - a2b1) = XN / D$$

$$y = (a1c2 - a2c1) / (a1b2 - a2b1) = YN / D$$

Esercizio: Sistema lineare

- Seguire i passi delineati nell'esempio precedente
  - Controllo errore → Valore di ritorno
  - Coefficienti → Parametri per valore
  - Soluzioni → Parametri per indirizzo
- Controllo errore
  - Ok, se X_N != 0, Y_N != 0, D != 0
  - Impossibile, se X_N != 0 oppure Y_N != 0, D == 0
  - Indeterminato, se X_N == 0 e Y_N == 0, D == 0

  →Tre possibili valori... un "enumerativo"!

Esercizio: Sistema lineare

- Interfaccia

Definisce un tipo che può assumere solo i valori specificati → i valori sono mappati su interi (da 0 in poi...)

```
typedef enum { ok , impossibile, indeterminato }
   TipoSistema;


TipoSistema sistema(int a1, int b1, int c1,
   int a2, int b2, int c2,
   float *x, float *y);
```

Esercizio: Sistema lineare

```
int main()
{
   TipoSistema tipoSistema;
   int a1, b1, c1, a2, b2, c2;  float x, y;
   printf("Inserire coefficienti eq. 1: ");
   scanf("%d %d %d%\n",&a1,&b1,&c1);
   printf("inserire coefficienti eq. 2: ");
   scanf("%d %d %d%\n",&a2,&b2,&c2);
   tipoSistema = sistema(a1, b1, c1, a2, b2, c2, &x, &y);
   switch (tipoSistema)
   {
        case ok: printf("%f %f\n", x, y);
                 break;
        case impossibile: printf("Sistema impossibile");
                 break;
        case indeterminato: printf("Sistema indeterminato");
                 break;
   }
}
```


16

Esercizio: Sistema lineare

```
TipoSistema sistema (int a1, int b1, int c1, int a2, int b2,
   int c2, float *x, float *y)
{
   int XN, YN, D;
   XN = c1*b2 - c2*b1;
   YN = a1*c2 - a2*c1;
   D = a1*b2 - a2*b1;
   if (D == 0)
   {
        if (XN == 0)&&(YN == 0) return indeterminato;
        else return impossibile;
   }
   else
   {
        *x = (float) (XN) / D;
        *y = (float) (YN) / D;
        return ok;
   }
}
```


17