Fondamenti di Informatica

Allievi Automatici
A.A. 2015-16

Tipi definiti dall'utente (typedef)
Tipi di dato astratti (TDA)

Tipi di dato

- Il **tipo** è un concetto astratto che esprime l'insieme dei valori e le operazioni permesse
- I tipi possono essere:
  - *predefiniti* (*built-in*) oppure *definiti dal programmatore*
  - *semplici* oppure *strutturati*
- Esempi:
  - *int, float, double, char, ecc.* (tipi semplici, predefiniti)
  - i tipi *enumerativi* (semplici, def. dal programmatore)
  - gli *array* e i *record* (strutturati, def. dal programmatore)

Definizione di nuovi tipi

- Esempio:

```
typedef int intero;
intero numero;
```

  è equivalente a:

```
int numero;
```

- È la definizione di un **sinonimo**
  - Il nuovo tipo **`intero`** eredita le caratteristiche (valori e operazioni) del tipo di partenza **`int`**

Definizione di tipi enumerativi

- Definizione di un nuovo tipo (enumerativo):

```
typedef enum { gen, feb, mar, apr, mag, giu,
  lug, ago, set, ott, nov, dic } tipo_mese;
```

- Dichiarazione di una **variabile** di tipo **`tipo_mese`**:

```
tipo_mese mese;
```

- È da notare la differenza con:

```
enum { ... } mese;
```

DOVE NON SI DEFINISCE UN TIPO NUOVO,
MA SOLO UNA NUOVA VARIABILE

Enumerazioni

- Normalmente, in C il valore intero 0 significa FALSO, e 1 significa VERO
- Il tipo LOGICO, o BOOLEAN:

```
typedef enum {falso, vero} boolean;
```

- Come già detto:
  - internamente le costanti dell'enumerazione sono rappresentate da numeri a partire da 0 in avanti, ma si può imporre diversamente:

```
typedef enum {lun = 1, mar, mer, gio,
ven, sab, dom} tipo_giorno;
```

  - sono definite le operazioni <, ==, >, ecc.

Costruttore di record (struct)

- Definiamo un tipo **`tipo_data`** per le date:

```
typedef struct {
        int giorno;
        tipo_mese mese;
        int anno;
} tipo_data;
```

- Possiamo poi dichiarare tre variabili:

```
tipo_data oggi, domani, dopodomani;
```

Costruttore di record (struct)

- Altro esempio:

```
typedef struct {
        char nome[20];
        char cognome[20];
        tipo_data nato_il;
        char nato_a[15];
        char codice_fiscale[16];
        int stipendio;
} dipendente;
```

Esempio

Si possono dichiarare le variabili:

```
dipendente persona1, persona2;
```

E calcolare un aumento di stipendio del 10% :

```
persona1.stipendio +=
   (persona1.stipendio * 10) / 100;

persona2.stipendio +=
   (persona2.stipendio * 10) / 100;
```

Altro esempio

```
typedef enum {disabled, enabled} Status;
typedef struct {
      int NumeroCanale;
      Status statoProgrammazione;
      float      LivelloLuminosita,
                 LivelloColore,
                 LivelloVolume;
} CanaliTV;
```

- Com'è fatto e a che cosa può servire il tipo CanaliTV?

Array

- Combinando typedef e costruttore di array:

```
typedef char venticaratteri[20];
```

si introduce un nuovo tipo, **venticaratteri**, il tipo degli array di 20 elementi di tipo **char**

- La dichiarazione

```
venticaratteri nome, cognome;
```

indica che **nome** e **cognome** sono di tipo **venticaratteri**, cioè due array di 20 **char**

Array e costruttore di tipo

- Possiamo costruire molti tipi diversi
  - typedef int iArray[20]; /* un array di 20 int */
  - typedef double dArray[30]; /* 30 double */
- Gli elementi di un array, inoltre, possono a loro volta essere di un "tipo array"
  - Dichiarazione alternativa per array multidimensionali

Matrici multidimensionali
(costruttori abbreviati)

- Introdurre il tipo delle matrici di 20×30 elementi:

```
typedef int Matrix2D[20][30];
```

- Dichiarare A come una variabile di tipo **`Matrix2D`**:

```
Matrix2D A;
```

- Introdurre il tipo delle matrici di 20×30×40 elementi:

```
typedef float Matrix3D[20][30][40];
```

- Per dichiarare B come variabile di tipo **`Matrix3D`**:

```
Matrix3D B;
```

- E così via ...

Array di array

```
typedef int Vettore[20];
typedef Vettore MatriceIntera20Per20[20];
                /*un array di 20 elementi di tipo Vettore*/

typedef int Matrice20Per20[20][20];
        /*è equivalente alle dichiarazioni precedenti*/

MatriceIntera20Per20 matrice;
int matrice[20][20];       /*dichiarazione abbreviata*/
```

Esercizio

Calcolo del perimetro di un poligono
Ci serve un modello!

- typedef struct { float x; float y; } punto;
- ~~typedef struct { punto p1; punto p2; } **lato**;~~ ?? **NO**

Usiamo l'array punto *N*-agono[*N*];

ESEMPIO (se N è 5):

```
typedef punto Pentagono[5];
Pentagono pn = {{0, 0}, {0.2, 1}, {1, 1}, {2, 0.5}, {1, -.3}};
        o, equivalentemente,
Pentagono pn = {0, 0, 0.2, 1, 1, 1, 2, 0.5, 1, -.3};
```

Esercizio

- Usiamo l'array punto *N*-agono[*N*];
  - Per calcolare il perimetro calcoliamo la somma delle distanze tra coppie di punti consecutivi
- N.B. Le coppie sono N
  - Dobbiamo ricordarci di calcolare anche la distanza tra il primo e l'ultimo punto
    - sono due punti non consecutivi nel vettore, ma rappresentano l'ultimo lato (quello che chiude la spezzata)

Soluzione

```
#include <math.h>
#define  N  5
.....
   int i = 0;
   punto pol[N];
   float perim = 0;
   while ( i < N-1 ) {
         perim += sqrt(    ( pol[i].x - pol[i+1].x ) *
                           ( pol[i].x - pol[i+1].x ) +
                           ( pol[i].y - pol[i+1].y ) *
                           ( pol[i].y - pol[i+1].y )   );
         i++;
   }
   perim += sqrt( ( pol[0].x - pol[N-1].x ) * ... + ... * ... );
```

Che cosa manca?

Ragioniamo ancora sui **modelli**

- Per costruire il modello del poligono abbiamo:
  - Scelto ***quali informazioni*** memorizzare
    - Le coordinate dei punti nell'ordine in cui definiscono i segmenti (non, ad esempio, le lunghezze dei lati e le misure degli angoli)
  - Scelto di ***quali astrazioni*** avvalerci per organizzarle
    - Un vettore di record (non, ad esempio, due vettori separati x[] e y[])
- Ma il modello non si esaurisce nella rappresentazione dell'oggetto
  - Si completa con la formulazione delle ***operazioni*** definite sull'oggetto ***in termini della rappresentazione scelta***
    - Calcolo del perimetro di un poligono → "operatore unario" **per(*pol*)**

Ragioniamo ancora sui **modelli**

- Da qui a creare il "tipo di dato" poligono il passo è breve
  - Struttura dei dati (formato e codifica)
  - Operazioni che a tali dati si applicano

vediamo un altro esempio

Rappresentazione di insiemi tramite array

- Vogliamo rappresentare **insiemi di numeri interi**
  - Gli insiemi non ammettono duplicati
  - Per semplicità, consideriamo solo i primi N interi

Possiamo dichiarare un array di N interi: `int insieme[N];`

o un tipo *tipo_insieme*:

```
typedef int tipo_insieme[N];
tipo_insieme insieme;
```

- Proposta:
  - un intero *i* appartiene all'insieme se e solo se nell'array l'elemento in posizione *i* ha valore *diverso da 0* (cioè vero)
    - L'insieme vuoto ∅ è rappresentato da un array di N zeri
    - Un insieme di n elementi ($0 \le n \le N$) è rappresentato da un array con esattamente n **uni** e N–n **zeri**

Rappresentazione di insiemi tramite array

- E per rappresentare **insiemi di lettere**? → N = 26 !!
  - Sfruttiamo l'ordinamento delle lettere per metterle in corrispondenza con gli interi nell'array [A→0, B→1, … Z→25]

```
typedef int alfabeto[26];
```

- ∅ o {}

```
alfabeto emptyset  = {0,0,0, … 0}   (26 zeri)
```

- {A}

```
alfabeto soloa     = {1,0,0, … 0}   (25 zeri)
```

- {A, Z}

```
alfabeto testacoda = {1,0 … 0,1}    (24 zeri)
```

- {A, C, F}

```
alfabeto éisièf    = {1,0,1,0,0,1,0 … 0}
```

Rappresentazione di insiemi tramite array

- Ragioniamo anche su come effettuare le operazioni base
  - N.B. Gli insiemi sono intrinsecamente ordinati
    - In virtù dell' associazione biunivoca tra indici ed elementi
- Siano dati i seguenti insiemi di lettere:

```
alfabeto S, T, U, I;
```

- Calcoliamo in `U` l'unione di `S` e `T` :

```
i = 0;
while( i < N ) {
   U[i] = S[i] || T[i]; /* assegna 0 o 1 */
   i = i + 1;
}
```

- Calcoliamo in `I` l'intersezione di `S` e `T` :

```
i = 0;
while( i < N ) {
   I[i] = S[i] && T[i]; /* assegna 0 o 1 */
   i = i + 1;
}
```

...e la differenza tra due insiemi?

Sono **modelli** che **definiscono** le operazioni base, blocchi di codice "**paradigmatici**", da ricordare e **riusare** quando serve fare tali operazioni

⇩

sottoprogrammi...