Tipo di dato

Definizione generale: **Insieme di valori e di operazioni ad esso applicabili**

Tipo astratto: conta la visione esterna, non la rappresentazione interna, ovvero il punto di vista di chi usa il tipo, non di chi lo realizza

Classificazione dei tipi di dato

- ✓ Tipi **semplici** (interi, caratteri, reali, ...)
- ✓ Tipi **predefiniti** o *built in* (nel linguaggio) (interi, caratteri, reali, ...)
- ✓ Tipi **definiti dall'utente**
- ✓ Tipi **strutturati** (array, ...)

⇨ in C: data, tabella non sono predefiniti: devono essere costruiti dal programmatore

⇨ in altri linguaggi "*special purpose*" potrebbero esserlo (e.g. una data nei fogli elettronici)

La dichiarazione dei dati, e quindi la dichiarazione del loro tipo, consente di conoscere:

- ☞ l'insieme dei valori ammissibili
- ☞ l'insieme delle operazioni applicabili
- ☞ la quantità di memoria necessaria
- ☞ gli eventuali errori d'uso.

TIPI SEMPLICI PREDEFINITI DEL C

Tipo semplice predefinito	*Alternative*
`char` **`signed char`** **`unsigned char`**	con eventuale qualificatore (**`signed`** o **`unsigned`**)
`int` **`signed short int`** **`signed int`** **`signed long int`** **`unsigned short int`** **`unsigned int`** **`unsigned long int`**	con eventuale qualificatore (**`signed`** o **`unsigned`** e **`short`** o **`long`**)
`float`	
`double`	con eventuale qualificatore **`long`**

Il tipo int

Operazioni *built-in* per dati di tipo **int**

=	Assegnamento di un valore **int** a una variabile **int**
+	Somma (tra **int** ha come risultato un **int**)
-	Sottrazione (tra **int** ha come risultato un **int**)
*****	Moltiplicazione (tra **int** ha come risultato un **int**)
/	Divisione con troncamento della parte non intera (risultato **int**)
%	Resto della divisione intera
==	Relazione di uguaglianza
!=	Relazione di diversità
<	Relazione “minore di”
>	Relazione “maggiore di”
<=	Relazione “minore o uguale a”
>=	Relazione “maggiore o uguale a”

I tipi **float** e **double**

Due diverse rappresentazioni:

– la normale **rappresentazione decimale**, o **in virgola fissa**:
  3.14
  1234.543328
  543.
  0.000076

– la **rappresentazione in virgola mobile** (*floating point*) con **mantissa** ed **esponente** (della base 10), separate dal carattere "**E**".
  1780000.0000023 può essere rappresentato in virgola mobile nei modi seguenti:
  178000.00000023E1
  1780000000023E-7
  1.780000000023E+6

Le notazioni sono interscambiabili e la macchina provvede automaticamente alle necessarie conversioni.

Spazio allocato (**float**) ≤ spazio allocato (**double**) ≤ spazio allocato (**long double**)

Operazioni *built-in* per dati di tipo **float** e **double**

=	Assegnamento
+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione (a risultato reale)
==	Relazione di uguaglianza
!=	Relazione di diversità
<	Relazione “minore di”
>	Relazione “maggiore di”
<=	Relazione “minore o uguale a”
>=	Relazione “maggiore o uguale a”

La standard library fornisce anche diverse funzioni matematiche predefinite: (sqrt, pow, exp, sin, cos, tan…) (per **double**).

Attenzione agli arrotondamenti:

(x/y) * y == x potrebbe risultare falsa !

Invece di scrivere:

if (x == y) …è meglio scrivere: **if** (abs (x – y) <= 0.000001) …

Il tipo **char**

L'insieme dei dati di tipo **char**, è l'insieme dei caratteri ASCII, e contiene tutte le lettere, le cifre e i simboli disponibili sulle normali tastiere.

La codifica ASCII consente la rappresentazione di ciascun carattere attraverso un opportuno valore intero.

La codifica ASCII definisce l'ordinamento dei valori: per una qualsiasi coppia di caratteri *x* e *y*, *x* < *y* se e solo se *x* precede *y* nell'elenco dei caratteri.

I caratteri di controllo:
\n = "a capo", \b = "backspace", \t = "horizontal tab",
\r = "carriage return", ETX, EOF, ….

Sono definite le operazioni di assegnamento (=), le operazioni aritmetiche (+, –, *, /, %) e quelle relazionali (==, !=, < ecc.)

Leggere una sequenza di caratteri conclusa dal carattere # e, per ciascun carattere stampare il corrispondente codice ASCII. Nel caso sia una lettera minuscola dell'alfabeto, trasformarla in lettera maiuscola.

```
/* Programma ManipolazioneCaratteri */
#include <stdio.h>
main()
{
    char C, CM;
    printf("Inserire un carattere – # per terminare il programma\n");
    scanf(" %c", &C);
/*NB lo spazio prima di %*/
    while (C != '#')
    {
        printf("Il codice ASCII del carattere %c è %d\n", C, C);
/* Se il carattere è una lettera minuscola */
        if (C >= 'a' && C <= 'z')
        {
/* La differenza 'a' – 'A' è lo scarto fra la rappresentazione ASCII delle
lettere maiuscole e minuscole dell'alfabeto */
        CM = C – ('a' –'A');
        printf("La lettera maiuscola per %c è %c e il suo codice ASCII è %d\n",
        C, CM, CM);
        }printf("Inserire un carattere – # per terminare il programma\n");
        scanf(" %c", &C);
    }
}
```

Compatibilità tra tipi

Espressioni che coinvolgono elementi eterogenei in tipo

Un'espressione aritmetica come **x + y** è caratterizzata dal valore e dal tipo del risultato.

Il tipo degli operandi condiziona l'operazione che deve essere eseguita: a operandi di tipo **int** si applica l'operazione di somma propria di tale tipo; diversa è l'operazione di somma che si applica a operandi di tipo **float** ecc.

Se **x** è di tipo **short** e **y** di tipo **int** è necessario convertire una delle due variabili per rendere omogenea l'espressione e applicare la corretta operazione: **x** viene temporaneamente convertita in **int** e la somma tra interi restituisce un risultato intero.

In generale si applicano le **Regole di conversione implicita**:

- ✓ ogni variabile di tipo **char** o **short** (incluse le rispettive versioni **signed** o **unsigned**) viene convertita in variabile di tipo **int**;
- ✓ se, dopo l'esecuzione del passo 1, l'espressione risulta ancora eterogenea rispetto al tipo degli operandi coinvolti, rispetto alla gerarchia

  int < **long** < **unsigned** < **unsigned long** < **float** < **double** < **long double**

  si converte temporaneamente l'operando di tipo inferiore facendolo divenire di tipo superiore;
- ✓ il risultato dell'espressione avrà tipo uguale a quello di più alto livello gerarchico.

Assegnamenti che coinvolgono elementi eterogenei in tipo

Le regole di conversione implicita esposte vengono utilizzate anche per la valutazione di assegnamenti tra variabili eterogenee in tipo:

```
Double d;
int i;
```

L'istruzione

```
d = i;
```

provoca una temporanea conversione del valore dell'intero `i` a **double** e successivamente l'assegnamento di tale valore **double** a `d`.

Invece:

```
i = d;
```

comporta invece, normalmente, una *perdita di informazione*. Il valore di `d` subisce infatti un *troncamento* alla parte intera con perdita della parte decimale.

E per creare tipi diversi da quelli semplici predefiniti?

Il costruttore typedef

```
typedef      int    anno;
```

Una volta definito e identificato un nuovo tipo ogni variabile può essere dichiarata di quel tipo come di ogni altro tipo già esistente:

```
anno   y;
```

NB: **`typedef`** non consente di costruire veri e propri nuovi tipi astratti (mancano le operazioni ...).

Ridefinizione

```
typedef    TipoEsistente     NuovoTipo;
```

TipoEsistente può essere sia un tipo *built-in* (predefinito), sia un tipo precedentemente definito:

```
typedef    int     tipo1;
typedef    char    tipo2;
typedef    tipo1   tipo3;
typedef    tipo2   tipo4;
```

Il costruttore enum — Enumerazione esplicita dei valori

```
typedef    enum    {lun, mar, mer, gio, ven, sab, dom}  GiornoDellaSettimana;
typedef    enum    {rosso, verde, giallo, arancio, marrone, nero}  colore;
typedef    enum    {Giovanni, Claudia, Carla, Simone, Serafino}  persone;
typedef    enum    {gen, feb, mar, apr, mag, giu, lug, ago, set, ott, nov, dic}
                   mese;
persone       individuo, individuo1, individuo2;

individuo = Giovanni;
……
if (individuo1 == individuo2) individuo = Claudia;
```

Alcune osservazioni:

- Spesso i valori del nuovo tipo sono rappresentati da nomi; però il compilatore associa a tali nomi un progressivo valore intero
  Per esempio, x di tipo `mese`:
  `gen` è in realtà `0`, `apr` è in realtà `3`, ecc.
- Operazioni applicabili: le stesse degli interi. Le seguenti operazioni:
  ```
  apr < giu
  rosso < arancio
  ```
  producono come risultato un intero diverso da 0 (valore logico "true"); mentre:
  ```
  dom < lun
  ```
  `Simone < Giovanni` producono 0 (valore "false").

Un importante caso particolare

```
typedef      enum  {false, true} boolean;
boolean      flag, ok;
```

`flag` e `ok` possono così essere definite come variabili in grado di assumere valore vero (`true`) o falso (`false`) durante l'esecuzione di un programma che le usi.

NB: non invertire l'ordine!