Tipi reali

I reali vengono rappresentati in virgola mobile (floating point).

- 3 tipi:
  `float`
  `double`
  `long double`
- **Intervallo di definizione:**

	sizeof	cifre significative	min esp.	max esp.
`float`	4	6	−37	38
`double`	8	15	−307	308
`long double`	12	18	−4931	4932

- Le grandezze precedenti dipendono dal compilatore e sono definite nel file `float.h`.
- Deve comunque valere la relazione:
  `sizeof(float)` $\leq$ `sizeof(double)` $\leq$ `sizeof(long double)`

Costanti: con punto decimale o notazione esponenziale

Esempio:

```
double x, y, z, w;
x = 123.45;
y = 0.0034;     /* oppure y = .0034 */
z = 34.5e+20;   /* oppure z = 34.5E+20 */
w = 5.3e-12;
```

- Nei programmi, per denotare una costante di tipo
  - `float`, si può aggiungere `f` o `F` finale
    Esempio: `float x = 2.3e5f;`
  - `long double`, si può aggiungere `L` o `l` finale
    Esempio: `long double x = 2.34567e520L;`

Operatori: come per gli interi (tranne "%")

Ingresso/uscita: tramite `printf` e `scanf`, con diversi specificatori di formato

Output con printf (per `float`):

- `%f` ... notazione in virgola fissa
  `%8.3f` ... 8 cifre complessive, di cui 3 cifre decimali

  Esempio:

```
float x = 123.45;
printf("|%f| |%8.3f| |%-8.3f|\n", x, x, x);
```

```
|123.449997| | 123.450| |123.450 |
```

- `%e` (oppure `%E`) ... notazione esponenziale
  `%10.3e` ... 10 cifre complessive, di cui 3 cifre decimali

  Esempio:

```
double x = 123.45;
printf("|%e| |%10.3e| |%-10.3e|\n", x, x, x);
```

```
|1.234500e+02| | 1.234e+02| |1.234e+02 |
```


Input con scanf (per `float`):
si può usare indifferentemente `%f` o `%e`.

Riassunto degli specificatori di formato per i tipi reali:

	float	double	long double
printf	`%f`, `%e`	`%f`, `%e`	`%Lf`, `%Le`
scanf	`%f`, `%e`	`%lf`, `%le`	`%Lf`, `%Le`

Conversioni di tipo

Situazioni in cui si hanno conversioni di tipo

- quando in un'espressione compaiono operandi di tipo diverso
- durante un'assegnamento `x = y`, quando il tipo di `y` è diverso da quello di `x`
- esplicitamente, tramite l'operatore di cast
- nel passaggio dei parametri a funzione (più avanti)
- attraverso il valore di ritorno di una funzione (più avanti)

Una conversione può o meno coinvolgere un cambiamento nella rappresentazione del valore.

da `short` a `long` (dimensioni diverse)
da `int` a `float` (anche se stessa dimensione)

Conversioni implicite tra operandi di tipo diverso nelle espressioni

Quando un'espressione del tipo x *op* y coinvolge operandi di tipo diverso, avviene una conversione implicita secondo le seguenti regole:

1. ogni valore di tipo `char` o `short` viene convertito in `int`
2. se dopo il passo 1. l'espressione è ancora eterogenea si converte l'operando di tipo inferiore facendolo divenire di tipo superiore secondo la seguente gerarchia:

`int` → `long` → `float` → `double` → `long double`

Esempio: `int x; double y;`
Nel calcolo di `(x+y)`:

1. `x` viene convertito in `double`
2. viene effettuata la somma tra valori di tipo `double`
3. il risultato è di tipo `double`

Conversioni nell' assegnamento

Si ha in `x = exp` quando i tipi di `x` e `exp` non coincidono.

- La conversione avviene sempre a favore del tipo della variabile a sinistra:

  se si tratta di una promozione non si ha perdita di informazione

  se si ha una retrocessione si può avere perdita di informazione

  Esempio:

```
int i;
float x = 2.3, y = 4.5;
i = x + y;
printf("%d", i); /* stampa 6 */
```

- Se la conversione non è possibile si ha errore.

Conversioni esplicite (operatore di cast)

Sintassi: `(tipo) espressione`

- Converte il valore di `espressione` nel corrispondente valore del `tipo` specificato.

  Esempio:

```
int somma, n;
float media;
...
media = somma / n;          /* divisione tra interi */
media = (float)somma / n;   /* divisione tra reali */
```

- L'operatore di cast "`(tipo)`" ha precedenza più alta degli operatori binari e associa da destra a sinistra. Dunque
  `(float) somma / n`
  equivale a
  `((float) somma) / n`

Input/output

- Come già detto, input e output non sono parte integrante del C
- L'interazione con l'ambiente è demandato alla libreria standard
  $\Longrightarrow$ un insieme di funzioni a uso dei programmi C
- La libreria `stdio.h` implementa un semplice modello di ingresso e uscita di dati testuali
- un testo è trattato come un successione (stream) di caratteri, ovvero
  $\Longrightarrow$ una sequenza di caratteri organizzata in righe, ciascuna terminata da "`\n`"
- al momento dell'esecuzione, al programma vengono connessi automaticamente 3 stream:
  - standard input: di solito la tastiera
  - standard output: di solito lo schermo
  - standard error: di solito lo schermo

Input/output (cont.)

- Compito della libreria è fare in modo che tutto il trattamento dei dati in ingresso e uscita si conformi a questo modello
  $\Longrightarrow$ il programmatore non si deve preoccupare di come ciò sia effettivamente realizzato
- Ogni volta che si effettua una operazione di lettura attraverso `getchar` viene acquisito il prossimo carattere dallo standard input e viene restituito il suo valore
  (analogamente per `scanf` che comporta l'acquisizione di uno o più caratteri a seconda delle specifiche di formato presenti ...)
- Ogni volta che si effettua una operazione di scrittura (attraverso `putchar` o `printf`) tutti i valori coinvolti vengono convertiti in sequenze di caratteri e quest'ultime vengo accodate allo standard output.
- Tipicamente il sistema operativo consente di redirigere gli stream standard, ad esempio su uno o più file.

Formattazione dell'output con `printf`

- Riepilogo specificatori di formato principali:
  - interi: `%d`, `%o`, `%u`, `%x`, `%X`
    per `short`: si antepone `h`
    per `long`: si antepone `l` (minuscola)
  - reali: `%e`, `%f`, `%g`
    per `double`: non si antepone nulla
    per `long double`: si antepone `L`
  - caratteri: `%c`
  - stringhe: `%s` (le vedremo più avanti)
  - puntatori: `%p` (li vedremo più avanti)
- Flag: messi subito dopo il "%"
  - "-": allinea a sinistra
  - altri flag (non ci interessano)
- Sequenze di escape: `\%`, `\'`, `\"`, `\\`, `\a`, `\b`, `\n`, `\t`, ...

Formattazione dell'input con `scanf`

- Specificatori di formato: come per l'output, tranne che per i reali
  - `double`: si antepone `l`
  - `long double`: si antepone `L`
- **Soppressione dell'input:** mettendo "*" subito dopo "%"
  Non ci deve essere un argomento corrispondente allo specificatore di formato.

  Esempio: Lettura di una data in formato gg/mm/aaaa oppure gg-mm-aaaa.

```
int g, m, a;
scanf("%d%*c%d%*c%d%*c", &g, &m, &a);
```

Espressioni booleane

- Come già sappiamo, il linguaggio deve consentire di descrivere espressioni booleane (guardie di condizionali e iterazione).
- In C non esiste un tipo Booleano $\Longrightarrow$ si usa il tipo **int** :

  falso $\Longleftrightarrow$ 0

  vero $\Longleftrightarrow$ 1 (in realtà qualsiasi valore diverso da 0)

  Esempio: $2 > 3$ ha valore 0 (ossia falso)

  $5 > 3$ ha valore 1 (ossia vero)

- **Operatori relazionali del C**
  - <, >, <=, >= (minore, maggiore, minore o uguale, maggiore o uguale) — priorità alta
  - ==, != (uguale, diverso) — priorità bassa

  Esempio: `temperatura <= 0` `velocita > velocita_max`
  `voto == 30` `anno != 2000`

Operatori logici

- In ordine di priorità:
  - ! (negazione) — priorità alta
  - && (congiunzione)
  - || (disgiunzione) — priorità bassa

Semantica:

a	b	!a	a && b	a || b
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

0 . . . falso

1 . . . vero (qualsiasi valore $\neq 0$)

Esempio:

`(a >= 10) && (a <= 20)` vero (1) se $10 \leq a \leq 20$

`(b <= -5) || (b >= 5)` vero se $|b| \geq 5$

- Le espressioni booleane vengono valutate da sinistra a destra:
  - con `&&`, appena uno degli operandi è falso, restituisce falso senza valutare il secondo operando
  - con `||`, appena uno degli operandi è vero, restituisce vero senza valutare il secondo operando
- **Priorità** tra operatori di diverso tipo:
  - not logico — priorità alta
  - aritmetici
  - relazionali
  - booleani (and e or logico) — priorità bassa

Esempio:

```
a+2 == 3*b || !trovato && c < a/3
```

è equivalente a

```
((a+2) == (3*b)) || ((!trovato) && (c < (a/3)))
```


Come va usato il codice dei caratteri

Esempio:

```
char x, y, z;
x = 'a';
y = 'd';
```

- Posso valutare $(x \leq y)$.
- Cosa mi dice? Se x precede y nell'ordine alfabetico.

Esempio:

```
char ch1 = 'a';
char ch2 = 'c';
char ch3 = (char) ((ch1 + ch2)/2);
printf("%c'', ch3);
```

- Cosa stampa? Stampa il carattere `'b'`.

Come va usato il codice dei caratteri

- Convertiamo una lettera minuscola in maiuscolo:

  Esempio:
  ```
  char lower = 'k';
  char upper = (char) (lower - 'a' + 'A');
  printf("%c", upper);
  ```

- Convertiamo un carattere numerico (una cifra) nell'intero corrispondente:

  Esempio:
  ```
  char ch1 = '9';
  int num = ch1-'0';
  ```

- Questi frammenti di programma sono completamente portabili (non dipendono dal codice usato per la rappresentazione dei caratteri).

Istruzione **if-else**

Sintassi:

```
if   (espressione)
     istruzione1
else istruzione2
```

- `espressione` è un'espressione booleana
- `istruzione1` rappresenta il ramo then (deve essere un'unica istruzione)
- `istruzione2` rappresenta il ramo else (deve essere un'unica istruzione)

Semantica:

1. viene prima valutata `espressione`
2. se `espressione` è vera viene eseguita `istruzione1`
   altrimenti (ovvero se `espressione` è falsa) viene eseguita `istruzione2`

```
int temperatura;

printf("Quanti gradi ci sono?  "); scanf("%d", &temperatura);
if (temperatura >= 25)
  printf("Fa caldo\n");
else
  printf("Si sta bene\n");

printf("Arrivederci\n");
```

```
=> Quanti gradi ci sono? 30 ↩
Fa caldo
Arrivederci
=>
```

```
=> Quanti gradi ci sono? 18 ↩
Si sta bene
Arrivederci
=>
```


Istruzione **if**

- È un'istruzione **if-else** in cui manca la parte else.

Sintassi:

```
if (espressione)
   istruzione
```

Semantica:

1. viene prima valutata `espressione`
2. se `espressione` è vera viene eseguita `istruzione` altrimenti non si fa alcunché

Esempio:

```
int temperatura;
scanf("%d", &temperatura);
if (temperatura >= 25)
  printf("Fa caldo\n");
printf("Arrivederci\n");
```

```
=> 18 ↩
Arrivederci
```

```
=> 30 ↩
Fa caldo
Arrivederci
```

Blocco

- La sintassi di **if-else** consente di avere un'unica istruzione nel ramo then (o nel ramo else).
- Se in un ramo vogliamo eseguire più istruzioni dobbiamo usare un blocco.

  Sintassi:

```
{
   istruzione-1
   ...
   istruzione-n
}
```

- Come già sappiamo e come rivedremo più avanti, un blocco può contenere anche dichiarazioni.

Esempio: Dati mese ed anno, calcolare mese ed anno del mese successivo.

```
int mese, anno, mesesucc, annosucc;

if (mese == 12)
{
  mesesucc = 1;
  annosucc = anno + 1;
}
else
{
  mesesucc = mese + 1;
  annosucc = anno;
}
```

If annidati (in cascata)

- Si hanno quando l'istruzione del ramo then o else è un'istruzione **if o if-else.**

  Esempio: Data una temperatura, stampare un messaggio secondo la seguente tabella:

temperatura t	messaggio
$30 < t$	Molto caldo
$20 < t \leq 30$	Caldo
$10 < t \leq 20$	Gradevole
$0 < t \leq 10$	Freddo
$t \leq 0$	Molto freddo

```
if (temperatura > 30)
  printf("Molto caldo\n");
else if (temperatura > 20)
  printf("Caldo\n");
else if (temperatura > 10)
  printf("Gradevole\n");
else if (temperatura > 0)
  printf("Freddo\n");
else
  printf("Molto freddo\n");
```


Osservazioni:

- si tratta di un'unica istruzione **if-else**

```
if (temperatura > 30)
  printf("Molto caldo\n");
else ...
```

- non serve che la seconda condizione sia composta

```
if (temperatura > 30) printf("Molto caldo\n");
else /* il valore di temperatura e' <= 30 */
    if (temperatura > 20)
```

  Non c'è bisogno di una congiunzione del tipo `(t <= 30) && (t > 20)`
  (analogamente per gli altri casi).

- **Attenzione**: il seguente codice

```
if (temperatura > 30) printf("Molto caldo\n");
if (temperatura > 20) printf("Caldo\n");
```

  ha ben altro significato (quale?)

Ambiguità dell'else

```
if (a >= 0) if (b >= 0) printf("b positivo");
else printf("???");
```

- `printf("???")` può essere la parte **else**
  - del primo **if** $\Longrightarrow$ `printf("a negativo");`
  - del secondo **if** $\Longrightarrow$ `printf("b negativo");`
- L'ambiguità sintattica si risolve considerando che un **else** fa sempre riferimento all'**if** più vicino, dunque

```
if (a > 0)
  if (b > 0)
    printf("b positivo");
  else
    printf("b negativo");
```

- Perché un **else** si riferisca ad un **if** precedente, bisogna inserire quest'ultimo in un blocco

```
if (a > 0)
  { if (b > 0) printf("b positivo"); }
else
  printf("a negativo");
```


Esercizio

Leggere un reale e stampare un messaggio secondo la seguente tabella:

gradi alcolici g	messaggio
$40 < g$	superalcolico
$20 < g \leq 40$	alcolico
$15 < g \leq 20$	vino liquoroso
$12 < g \leq 15$	vino forte
$10.5 < g \leq 12$	vino normale
$g \leq 10.5$	vino leggero

Esempio: Dati tre valori che rappresentano le lunghezze dei lati di un triangolo, stabilire se si tratti di un triangolo equilatero, isoscele o scaleno.

Algoritmo:
```
determina tipo di triangolo
    leggi i tre lati
    confronta i lati a coppie, fin quando non
      hai raccolto una quantità di informazioni
      sufficiente a prendere la decisione
    stampa il risultato
```



```
main()   {
float primo, secondo, terzo;

printf("Lunghezze lati triangolo ?  ");
scanf("%f%f%f", &primo, &secondo, &terzo);

if (primo == secondo) {
  if (secondo == terzo)
    printf("Equilatero\n");
  else
    printf("Isoscele\n");
}
else {
  if (secondo == terzo)
    printf("Isoscele\n");
  else if (primo == terzo)
    printf("Isoscele\n");
  else
    printf("Scaleno\n");
}     }
```

Esercizio

Risolvere il problema del triangolo utilizzando il seguente algoritmo:

```
Algoritmo: determina tipo di triangolo con conteggio
leggi i tre lati
confronta i lati a coppie contando
  quante coppie sono uguali
if le coppie uguali sono 0
  è scaleno
  else if le coppie uguali sono 1
    è isoscele
    else è equilatero
```