## Gnocchi in brodo (Pellegrino Artusi, 1891)

*... è una minestra da farsene onore; ma se non volete consumare appositamente per lei un petto di pollastra o di cappone, aspettate che vi capiti d'occasione. Cuocete nell'acqua, o meglio a vapore, grammi 200 di patate grosse e farinacee e passatele per istaccio. A queste unite il petto di pollo lesso tritato finissimo colla lunetta, grammi 40 di parmigiano grattato, due rossi d'uovo, sale quanto basta e odore di noce moscata. Mescolate e versate il composto sulla spianatoia sopra a grammi 30 o 40 (che tanti devono bastare) di farina per legarlo, e poterlo tirare a bastoncini grossi quanto il dito mignolo. Tagliate questi a tocchetti e gettateli nel brodo bollente ove una cottura di cinque o sei minuti sarà sufficiente. Questa dose potrà bastare per sette od otto persone. Se il petto di pollo è grosso, due soli rossi non saranno sufficienti...*

La ricetta è piacevole da leggere, ma non è facile da seguire.





## Gnocchi in brodo (cont.)

DOSE: 7-8 persone
INGREDIENTI: 200 di patate grosse e farinacee, 40 g parmigiano, 2 rossi di uovo, sale e noce moscata q.b., un petto di pollo, 30-40 g farina

- Cuocete nell'acqua o a vapore le patate
- passatele al setaccio
- unite pollo lesso tritato, parmigiano, uova, sale e noce moscata
- versate il composto sulla spianatoia, mescolandolo con la farina
- tirare il composto a bastoncini dello spessore del dito mignolo
- tagliare i bastoncini a tocchetti
- gettare i tocchetti nell'acqua bollente e cuocerli per 5-6 minuti.

La ricetta non è altrettanto piacevole da leggere, ma è facile da seguire, anche se presenta delle ambiguità.

# Introduzione al linguaggio C

- Abbiamo già visto come un programma non sia altro che un algoritmo codificato in un linguaggio di programmazione.
- Problema: quale linguaggio scegliere per la codifica di un algoritmo?
  - Il linguaggio naturale sarebbe facilmente comprensibile ma non è eseguibile da una macchina.
  - Il linguaggio macchina è eseguibile ma di difficile comprensione.
- Due requisiti fondamentali di un qualsiasi linguaggio per la descrizione di algoritmi:
  - deve essere preciso per non lasciare adito a dubbi interpretativi
  - deve essere sintetico per non rendere difficile la comprensione dei programmi.





- Il linguaggio naturale e il linguaggio macchina si collocano in posizioni opposte, soddisfacendo uno solo dei requisiti.
- I linguaggi di programmazione ad alto livello sono progettati proprio per colmare tale divario.
  $\Longrightarrow$ sono linguaggi adatti a codificare algoritmi pur rimanendo comprensibili.
- La fatica di tradurre un programma nel linguaggio macchina è affidata a particolari programmi, i compilatori, che traducono programmi scritti nel linguaggio di più alto livello in programmi equivalenti nel linguaggio macchina.

- Vedremo il cosidetto ANSI C (standard del 1989, con successive aggiunte)
- Il primo programma C: ciao mondo

```
#include <stdio.h>
main()
  /* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
}
```

- Questo programma stampa sullo schermo una riga di testo:

```
Ciao mondo!
>
```

- Vediamo in dettaglio ogni riga del programma.





```
/* Stampa un messaggio sullo schermo.  */
```

- testo racchiuso tra "/*" e "*/" è un commento
- i commenti sono a "uso umano", cioè servono a chi scrive o legge il programma, per renderlo più comprensibile
- il compilatore ignora i commenti
- attenzione a non dimenticare di chiudere i commenti con */, altrimenti tutto il resto del programma viene ignorato

`main()`

- è una parte presente in tutti i programmi C
- le parentesi "`(`" e "`)`" dopo main indicano che main è una funzione
- i programmi C sono composti da una o più funzioni, tra le quali ci deve essere la funzione `main`, che è quella principale
  $\Longrightarrow$ `main` è una funzione speciale, perché l'esecuzione del programma comincia l'esecuzione all'inizio di `main`
- la parentesi "`{`" apre il corpo della funzione e "`}`" lo chiude
  - la coppia di parentesi e la parte racchiusa da esse costituiscono un blocco
  - il corpo della funzione contiene le istruzioni (e dichiarazioni) che costituiscono la funzione





`printf("Ciao mondo!\n");`

- è un'istruzione semplice (ordina al computer di eseguire un'azione) in questo caso visualizzare (stampare) sullo schermo la sequenza di caratteri tra apici
- ogni istruzione semplice deve terminare con "`;`"
- oltre alle istruzioni semplici, esistono anche istruzioni composte (che non devono necessariamente terminare con "`;`")
- la parte racchiusa in una coppia di doppi apici è una stringa (di caratteri)
- "`\n`" non viene visualizzato sullo schermo, ma provoca la stampa di un carattere di fine riga
  - "`\`" è un carattere di escape e, insieme al carattere che lo segue, assume un significato particolare (sequenza di escape)
- in realtà anche `printf` è una funzione, e l'istruzione di sopra è un'attivazione di funzione (le vedremo più avanti)

`#include <stdio.h>`

- è una direttiva di compilazione
- viene interpretata dal compilatore durante la compilazione
- la direttiva "`#include`" dice al compilatore di includere il contenuto di un file nel punto corrente
- `<stdio.h>` è un file che contiene i riferimenti alla libreria standard di input/output (dove è definita la funzione `printf`)
- il linguaggio C non prevede istruzioni esplicite di input/output. Queste operazioni sono definite tramite funzioni nella libreria standard di input/output.

**Note:**

- è importante distinguere i caratteri maiuscoli da quelli minuscoli `Main`, `MAIN`, `Printf`, `PRINTF` non andrebbero bene
- si è usata l'indentazione per mettere in evidenza la struttura del programma ▸)





## Alcune varianti del programma

```
#include <stdio.h>
main()
  /* Stampa un messaggio sullo schermo. */
{
  printf("Ciao");
  printf(" mondo!\n");
}
```

- produce lo stesso effetto del programma precedente
- la seconda invocazione di `printf` incomincia a stampare dal punto in cui aveva smesso la prima
- Cosa viene stampato se usiamo

```
printf("Ciao");
printf("mondo!\n");
```

```
printf("Ciao\n");
printf("mondo!\n");
```

# Un altro programma: area di un rettangolo

```
#include <stdio.h>

main() {
  int base;
  int altezza;
  int area;

  base = 3;
  altezza = 4;
  area = base * altezza;

  printf("Area: %d\n", area);
}
```

Quando viene eseguito stampa:

```
Area: 12
>
```





# Le variabili (di programma)

Lo stato rappresenta il contenuto (modificabile) della memoria.

Una posizione della memoria, destinata a contenere dati, si identifica con una variabile Le variabili dunque rappresentano, nei programmi, le associazioni (modificabili) dello stato
$\Longrightarrow$ cf. $x \rightsquigarrow val$ nello pseudo-linguaggio

Una variabile è caratterizzata dalle seguenti proprietà:

1. nome: serve a identificarla — esempio: `area`
2. valore: valore associato allo stato corrente — Esempio: 4 (può cambiare durante l'esecuzione)
3. tipo: specifica l'insieme dei possibili valori assunti durante l'esecuzione — Esempio: `int` (numeri interi) e implicitamente le operazioni possibili.
4. indirizzo: della cella di memoria a partire dal quale è memorizzato il valore.

Nome, tipo e indirizzo non possono cambiare durante l'esecuzione.

## Le variabili (cont.)

- Il nome di una variabile è un identificatore C
  $\Longrightarrow$ sequenza di lettere, cifre, e _ che inizia con una lettera o con _
  Esempio: `Numero_elementi`, `x1`, ma non `1_posto`
  - può avere lunghezza qualsiasi, ma solo i primi 31 caratteri sono significativi
  - lettere minuscole e maiuscole sono considerate distinte
- Ad ogni variabile è associata una cella di memoria o più celle consecutive, a seconda del suo tipo. Il suo indirizzo è quello della prima cella.
- Analogia con una scatola di scarpe etichettata in uno scaffale
  - nome $\Longrightarrow$ etichetta
  - valore $\Longrightarrow$ scarpa che c'è nella scatola
  - tipo $\Longrightarrow$ capienza (che tipo di scarpe ci metto dentro)
  - indirizzo $\Longrightarrow$ posizione nello scaffale (la scatola è incollata)

  **N.B.**
  - non tutte le variabili sono denotate da un identificatore (strut. din.)
  - non tutti gli identificatori sono identificatori di variabile (ad es. funzioni, tipi, parole riservate, ...)





## Area del rettangolo

- `int base;` — è una dichiarazione di variabile
  - viene creata la scatola e incollata allo scaffale
  - ha tipo `int` $\Longrightarrow$ può contenere interi
  - ha nome `base`
  - ha un indirizzo (posizione nello scaffale), che è quello della cella di memoria associata alla variabile
  - ha un valore iniziale, che però non è significativo (è casuale)
    $\Longrightarrow$ la scatola viene creata piena, però con una scarpa scelta a caso, ovvero
    $\Longrightarrow$ l'associazione nello stato è del tipo nome $\rightsquigarrow$ ?

- `int altezza;`
  `int area;`
  $\Longrightarrow$ come per `base`

# Variabili numeriche

## Variabili intere

- per dichiarare variabili intere si può usare il tipo `int`
- i valori di tipo `int` sono rappresentati in C con almeno 16 bit
- il numero effettivo di bit dipende dal compilatore
  **Esempio:** 32 bit per il compilatore gcc (usato in ambiente Unix)
- in C esistono altri tipi per variabili intere (`short`, `long`) — li vedremo più avanti

## Variabili reali

- per dichiarare variabili reali si può usare il tipo `float`
  **Esempio:** `float temperatura;`





# Area del rettangolo

`base = 3;`
è un'istruzione di assegnamento (come nello pseudo-linguaggio)

- in C l'operatore di assegnamento è denotato dal simbolo "="
- come già sappiamo, l'effetto è di modificare una associazione nello stato
  $\Longrightarrow$ in questo caso il valore 3 viene associato a base, come?
  $\Longrightarrow$ il nuovo valore viene scritto nella spazio associato alla variabile
- a questo punto la variabile `base` ha un valore significativo
  $\Longrightarrow$ da base $\rightsquigarrow$ ? a base $\rightsquigarrow$ 3

`altezza = 4;` $\Longrightarrow$ come sopra

`area = base * altezza;`
a destra di "=" possono comparire espressioni $\Longrightarrow$ il valore assegnato è quello dell'espressione calcolata nello stato corrente

- una variabile all'interno di una espressione sta per il valore ad essa associato in quel momento (cf. pseudo-linguaggio)

Nota: operatori aritmetici tra interi del C sono: +, -, *, /, %, ...

## Area del rettangolo

```
printf("Area: %d\n", area);
```

- è un'istruzione di stampa
- il primo argomento è la stringa di formato che può contenere specificatori di formato
- lo specificatore di formato `%d` indica che deve essere stampato un intero in notazione decimale (`d` per decimal)
- ad ogni specificatore di formato nella stringa deve corrispondere un valore che deve seguire la stringa di formato tra gli argomenti di `printf`

  **Esempio:** `printf(" %d %d··· %d", i1, i2, ..., in );`
- nel caso di `printf("Ciao mondo!\n");` la stringa di formato non conteneva specificatori e quindi non vi erano altri argomenti.





## Struttura dei programmi C

- Nel semplice programma che abbiamo appena analizzato possiamo già vedere la struttura generale di un programma C.

```
/* DIRETTIVE DI COMPILAZIONE */
#include <stdio.h>
 main() {

 /* PARTE DICHIARATIVA */
  int base;
  int altezza;
  int area;

 /* PARTE ESECUTIVA */
  base = 3;
  altezza = 4;
  area = base * altezza;
  printf("Area: %d\n", area);
}
```

Un programma C deve contenere nell'ordine:

- Una parte contenente direttive per il compilatore. Nel nostro programma la direttiva

```
#include <stdio.h>
```

- l'identificatore predefinito `main` seguito dalle parentesi `()`.
- due parti racchiuse tra parentesi graffe
  - la parte dichiarativa. Nell'esempio:

```
int base;
int altezza;
int area;
```

  - la parte esecutiva. Nell'esempio:

```
base = 3;
altezza = 4;
area = base * altezza;
printf("Area: %d\n", area);
```





## La parte dichiarativa

- È posta prima della codifica dell'algoritmo e obbliga il programmatore a dichiarare i nomi simbolici che saranno presenti nello stato e di cui farà uso nella parte esecutiva. Contiene i seguenti elementi:
  - la sezione delle dichiarazioni di costanti
  - la sezione delle dichiarazioni di variabili;
- Le dichiarazioni: rendono più pesante la fase di costruzione dei programmi, ma consentono di individuare e segnalare errori in fase di compilazione.

  **Esempio:**

```
int x;
int alfa;
alfa = 0;
x=alfa;
alba=alfa+1;
```

- Il compilatore vede `alba` come variabile non dichiarata.

## Dichiarazioni di variabili

- Abbiamo già visto esempi di dichiarazioni di variabili.

```
float x;
int base;
int altezza;
```

- Ad ogni variabile viene attribuito, al momento della dichiarazione, un tipo
  $\Longrightarrow$ specifica l'insieme dei valori che la variabile può assumere
- La dichiarazione può anche attribuire un valore iniziale alla variabile (inizializzazione)

```
int x = 0;
```

- Variabili dello stesso tipo possono essere dichiarate contemporaneamente

```
int base, altezza, area;
```

(ma inizializzate singolarmente)

**Esempio:** `int x, y, z=0;` solo `z` è inizializzata a `0`.





## Dichiarazioni di costanti (variabili *read-only*)

- Una dichiarazione di costante crea un'associazione non modificabile
  $\Longrightarrow$ associa in modo permanente un valore ad un identificatore.

  **Esempio:**

```
const float PiGreco=3.14;
const int N=100;
```

- L'associazione tra il nome `PiGreco` ed il valore `3.14` non può essere modificata durante l'esecuzione.
- Come per le dichiarazioni di variabili, più costanti dello stesso tipo possono essere dichiarate insieme

  **Esempio:**

```
const float PiGreco=3.14, e=2.718;
const int N=100, M=200;
```

- **N.B.** cosa succede se si modifica una costante non è specificato dallo standard ANSI C, dipende dal compilatore.

## Uso di costanti

- Con la dichiarazione `const float PiGreco=3.14;`
  l'istruzione
  `AreaCerchio=PiGreco*RaggioCerchio*RaggioCerchio;`
  è equivalente a
  `AreaCerchio=3.14*RaggioCerchio*RaggioCerchio`
- Maggiore leggibilità dei programmi, dovuta all'uso di nomi simbolici
- Maggiore adattabilità dei programmi che usano costanti

  **Esempio:**
  Per aumentare la precisione, basta cambiare la dichiarazione in
  `const float PiGreco = 3.1415;`

  Senza l'uso della costante si dovrebbero rimpiazzare nel codice tutte le occorrenze di `3.14` in `3.1415` ...





## Area di un rettangolo di dimensioni lette da tastiera

```
#include <stdio.h>

main()
{
  int base, altezza, area;

  printf("Immetti base del rettangolo e premi INVIO\n");
  scanf("%d", &base);
  printf("Immetti altezza del rettangolo e premi INVIO\n");
  scanf("%d", &altezza);

  area = base * altezza;

  printf("Area: %d\n", area);
}
```

## Nuova istruzione: `scanf("%d", &base);`

- `scanf` è la funzione duale di `printf`
- legge da input (tastiera) un valore intero e lo assegna alla variabile `base`
- `"%d"` è la stringa di controllo del formato (in questo caso viene letto un intero in formato decimale)
- "`&`" è l'operatore di indirizzo
  - `&base` indica (l'indirizzo del)la locazione di memoria associata a `base`
  - `scanf` memorizza in tale locazione il valore letto
- quando viene eseguita `scanf` il programma si mette in attesa che l'utente immetta un valore. Quando l'utente digita Invio
  1. la sequenza di caratteri immessa viene convertita in un intero (formato `%d`) e
  2. l'intero ottenuto viene assegnato alla variabile `base` (viene cioè scritto nella/e cella/e di memoria a partire dall'indirizzo passato a `scanf`)

**N.B.** il precedente valore della variabile `base` va perduto (cf. Input(base) nello pseudo-linguaggio.)





## Esempio di esecuzione

- Vediamo cosa avviene durante l'esecuzione (indichiamo in rosso ciò che l'utente digita e in particolare con $\hookleftarrow$ il tasto Invio).

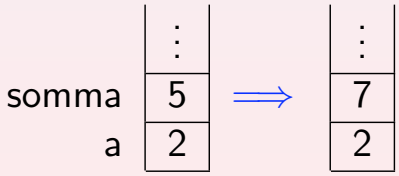
```
Immetti base del rettangolo e premi INVIO
5 ↩
Immetti altezza del rettangolo e premi INVIO
4 ↩
Area: 20
```

# Assegnamento

- Ricordiamo che l'esecuzione di `x = exp` corrisponde a:
  1. valutare il valore dell'espressione `exp` a destra di "=" (usando i valori correnti delle variabili);
  2. assegnare poi tale valore alla variabile `x` a sinistra di "=".

**Esempio:**

```
somma = 5;
a = 2;
somma = somma + a;
```








**Esempio:**

| | a | b |
|---|---|---|
| `int a, b;` | ? | ? |
| `a = 2;` | 2 | ? |
| `b = 3;` | 2 | 3 |
| `a = b;` | 3 | 3 |
| `a = a + b;` | 6 | 3 |
| `b = a + b;` | 6 | 9 |

# Osservazioni sull'assegnamento

- **Attenzione:** A sinistra di "=" ci deve essere un identificatore di variabile

  $\Longrightarrow$ denota la corrispondente associazione modificabile nello stato.

  **Esempio:** Quali istruzioni sono corrette e quali no?

  `a = a;` SI corretta (anche se poco significativa . . . )

  `a = 2 * a;` SI corretta (il valore associato ad `a` si raddoppia)

  `5 = a;` NO, `5` non denota una associazione modificabile nello stato ma un valore costante

  `a + b = c;` NO, `a+b` è un'espressione, non una variabile!