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 che abbiamo brevemente illustrato 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 gap.
  $\Longrightarrow$ 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.

Codifica in C

Essenzialmente ogni volta che andiamo a codificare un algoritmo come programma C dovremo specificare:

- Come leggere i dati di ingresso su cui lavorare: da tastiera, file o altro
- Come rappresentare i dati in ingresso e intermedi
- Quali passi devono essere fatti per realizzare l'algoritmo che abbiamo in mente
- Come fornire il risultato: su schermo, file o altro

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

```
#include <stdio.h>
int main()
  /* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

- 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 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

`int main()`

- è una parte presente in tutti i programmi C
- le parentesi "`(`" e "`)`" dopo main indicano che main è una funzione che non ha argomenti (equivalente a `int main(void)`) e restituisce un intero `int`,
- i programmi C sono composti da una o più funzioni, tra le quali ci deve essere la funzione `main`
  $\Longrightarrow$ `main` è una funzione speciale, perché l'esecuzione del programma incomincia con l'esecuzione 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
- in realtà anche `printf` è una funzione, e l'istruzione di sopra è un'attivazione di funzione (le vedremo più avanti)

```
return 0;
```

- Questa istruzione specifica che la funzione è terminata e fornisce il valore che deve essere restituito a chi ha richiesto l'esecuzione del programma C (ad esempio la shell)
- Per convenzione 0 (o, meglio, il valore predefinito EXIT_SUCCESS) significa tutto ok, mentre un qualsiasi altro valore indica un errore.

`#include <stdio.h>`

- è una direttiva di compilazione al preprocessore un programma di manipolazione testuale che elabora il sorgente prima del compilatore C.
- 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 miniscoli `Main`, `MAIN`, `Printf`, `PRINTF` non andrebbero bene
- si è usata l'indentazione per mettere in evidenza la struttura del programma (ma non é essenziale per la semantica come in Python)

Alcune varianti del programma

```
#include <stdio.h>
int main()
  /* Stampa un messaggio sullo schermo. */
{
  printf("Ciao");
  printf(" mondo!\n");
  return 0;
}
```

- 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>

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

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

  printf("Area: %d\n", area);
  return 0;
}
```

Quando viene eseguito stampa:

```
Area: 12
>
```

Il concetto di stato

Cosa si intende con stato?

- una particolare rappresentazione delle informazioni di una macchina, che in qualche modo "memorizza" le condizioni in cui si trova, e che cambia nel tempo passando da una configurazione all'altra per effetto delle istruzioni di un programma.
- Ad esempio, in un dato programma lo stato può rappresentare la posizione dell'ascensore ad un certo piano di un edificio. In base allo stato si determina il modo in cui l'ascensore si deve muovere per raggiungere il terzo piano.
- Un algoritmo è una sequenza di passi elementari che, se intrapresi da un esecutore, comportano ripetute modifiche dello stato iniziale fino al raggiungimento dello stato finale desiderato.
- Le azioni di base che l'esecutore è in grado di effettuare devono dunque prevedere, tra le altre, azioni che hanno come effetto cambiamenti dello stato.

Un'astrazione dello stato

Possiamo immaginarci lo stato nel sequente modo.

Stato

Uno stato è un insieme di associazioni tra nomi simbolici e valori che sono attivi in un dato momento di esecuzione del programma.

In uno stato, ad ogni nome simbolico è associato al più un valore.

Rappresentiamo una associazione tra il nome simbolico x ed il valore val con la notazione

$$x \rightsquigarrow val$$

Lo stato: esempi

Stati corretti

- $\{$nome $\rightsquigarrow$ Antonio, cognome $\rightsquigarrow$ Rossi, eta' $\rightsquigarrow$ 25$\}$
- $\{$importo $\rightsquigarrow$ \$1650, tasso $\rightsquigarrow$ 10%, interesse $\rightsquigarrow$ \$165 $\}$
- $\{$a $\rightsquigarrow$ 25, b $\rightsquigarrow$ 3, c $\rightsquigarrow$ 50$\}$

Stati non corretti

- $\{$nome $\rightsquigarrow$ Antonio, nome $\rightsquigarrow$ Paolo, eta' $\rightsquigarrow$ 25$\}$
- $\{$b $\rightsquigarrow$ 45, a $\rightsquigarrow$ 150, b $\rightsquigarrow$ 10$\}$

Le variabili

Una variabile è un nome simbolico per un valore;

Il C fornisce la possibilita' di definire variabili per contenere i valori necessari al calcolo

Una variabile è caratterizzata dalle seguenti proprietà:

1. nome: serve a identificarla — esempio: `area`
2. valore: valore associato nello stato corrente — Esempio: 4 (può cambiare durante l'esecuzione)
3. tipo: specifica l'insieme dei possibili valori — Esempio: `int` (numeri interi)
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
  - 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

- `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 `double`
  Esempio: `double temperatura;`

Area del rettangolo

`base = 3;`
è un'istruzione di assegnamento

- in C l'operatore di assegnamento è denotato dal simbolo "="
- l'effetto è di modificare un' 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

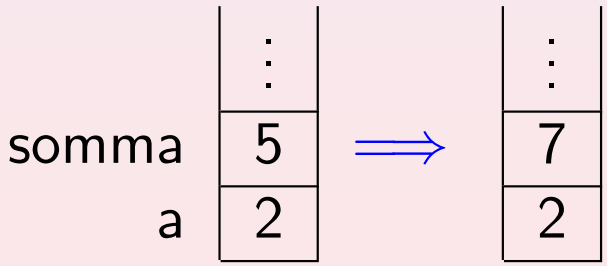
Nota: operatori aritmetici tra interi del C `+`, `-`, `*`, `/`, `%`, ...

Approfondimento sull' Assegnamento

- 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` viene raddoppiato)

  `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!

Esempio: Scambio del valore di due variabili:

Il seguente codice non funziona (si perde il valore di a):

```
a = b; b = a;
```

prima:

	⋮
a	5
b	8

dopo:

	⋮
a	8
b	5

$\Longrightarrow$ prima di eseguire `a = b` bisogna copiare il valore di a in una variabile temporanea per poterla poi assegnare a b:

	a	b	temp
	5	8	?
`temp = a;`	5	8	5
`a = b;`	8	8	5
`b = temp;`	8	5	5

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>
int main() {

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

 /* PARTE ESECUTIVA */
  base = 3;
  altezza = 4;
  area = base * altezza;
  printf("Area: %d\n", area);
  return 0;
}
```

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` preceduto da `int` e 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);
return 0;
```

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 variabili;
  - la sezione delle dichiarazioni di costanti.
- 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;
```

- Nell'ultima linea abbiamo erroneamente scambiato una **b** con una **f**
  $\Longrightarrow$ il compilatore individua `alba` come variabile non dichiarata.

Dichiarazioni di variabili

- Abbiamo già visto esempi di dichiarazioni di variabili.

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

- **N.B.** cosa succede quando si modifica una variabile read-only non è specificato dallo standard ANSI C, dipende dal compilatore.

Uso di costanti

- Con la dichiarazione `const double 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 double 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 con dimensioni reali lette da tastiera

```
#include <stdio.h>

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

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

  area = base * altezza;

  printf("Area: %f\n", area);
  return 0;
}
```

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

- `scanf` è la funzione duale di `printf`
- legge da input (tastiera) un valore reale e lo assegna alla variabile `base`
- `"%lf"` è la stringa di controllo del formato (in questo caso viene letto un numero reale 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 reale (formato `%lf`) e
  2. il valore reale 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.

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
5.6 ↩
Immetti altezza del rettangolo e premi INVIO
4.3 ↩
Area: 24.080000
```