Introduzione al Linguaggio C

Informatica B a.a. 2021 / 2022

Francesco Trovò

francesco1.trovo@polimi.it

Intermezzo: Cosa Farebbe Questo Algoritmo?

1. Alzatevi tutti in piedi
2. Ognuno di voi vale 1
3. **Ripeti:** Ciascuno cerca un compagno/a ancora in piedi
4. **Se** non avete trovato un compagno, il vostro valore non cambia e dovete restare in piedi
5. **Altrimenti** ogni coppia somma i loro valori, il risultato è il nuovo valore di ciascuno
6. Uno dei due si deve sedere e l'altro deve restare in piedi
7. Ricominciate dal punto 3, **finchè** non resta in piedi una sola persona in tutta la stanza
8. Il valore dell'ultima persona rimasta in piedi è:

il numero di persone presenti nella stanza

Linguaggi di Programmazione

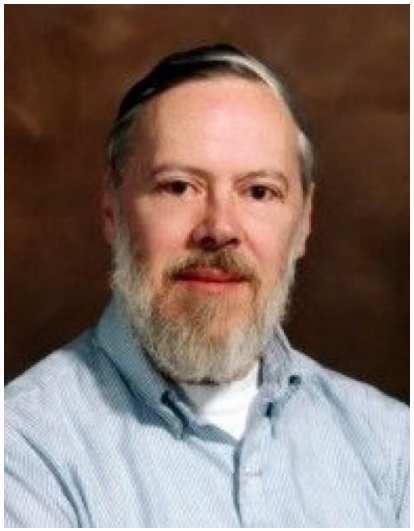
Programmazione a Basso / Alto Livello

- **Linguaggio macchina**: poche istruzioni, difficile codificare algoritmi e interpretare il codice:
  - linguaggio preciso
  - completo controllo delle risorse
- **Linguaggi di alto livello**: linguaggi più comprensibili per l'uomo:
  - linguaggio preciso e sintetico
  - riferimenti simbolici
  - esprimere istruzioni in linguaggio vicino a quello naturale
- La traduzione dal linguaggio ad alto livello al linguaggio macchina è eseguita da un altro programma, il **compilatore**

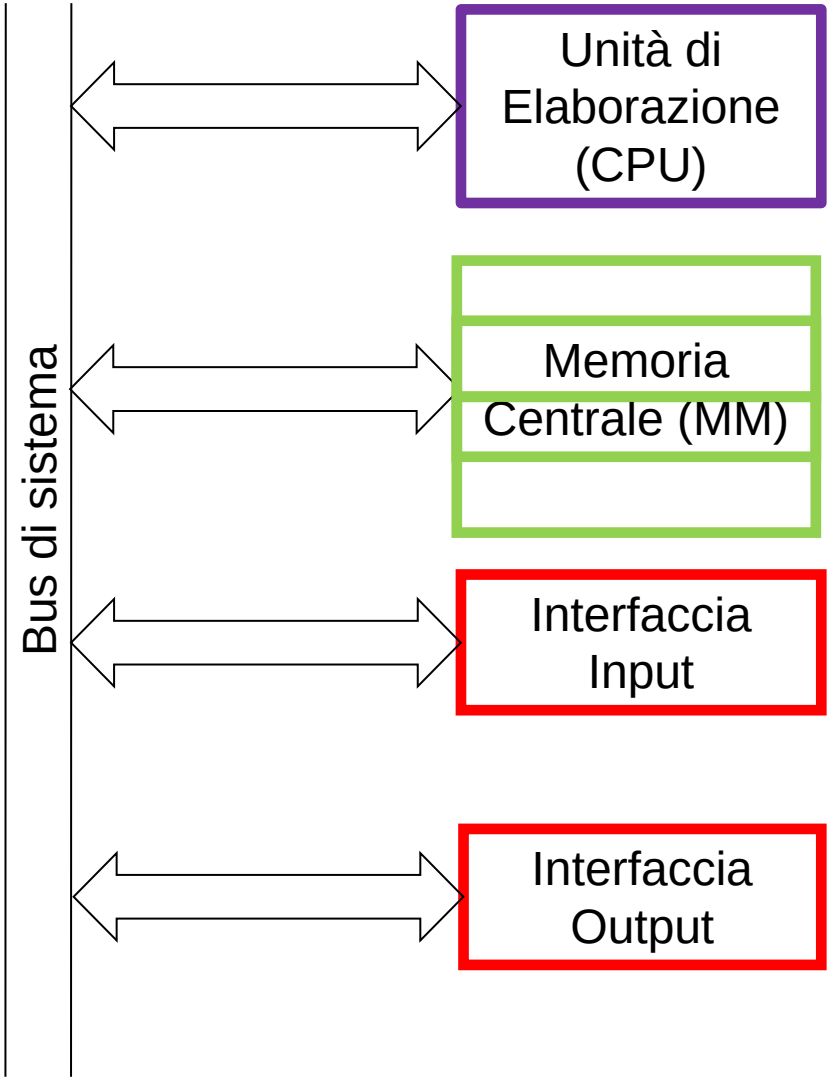
Il Linguaggio C

Il Linguaggio C: Qualche Informazione Storica

- Nasce nel 1972 da Dennis Ritchie come un linguaggio ad alto livello per la scrittura di sistemi operativi
  - Utilizzato per riscrivere UNIX
- Negli anni ‘80 si diffondono versioni del compilatore di linguaggio C per diverse architetture
- Nel 1983, l’ANSI (American National Standards Institute) lavora ad una versione standard del C, e anche recentemente con C99 e C11
- Ad oggi, gran parte dei sistemi operativi è ancora scritta in C (o C++)



La Macchina C



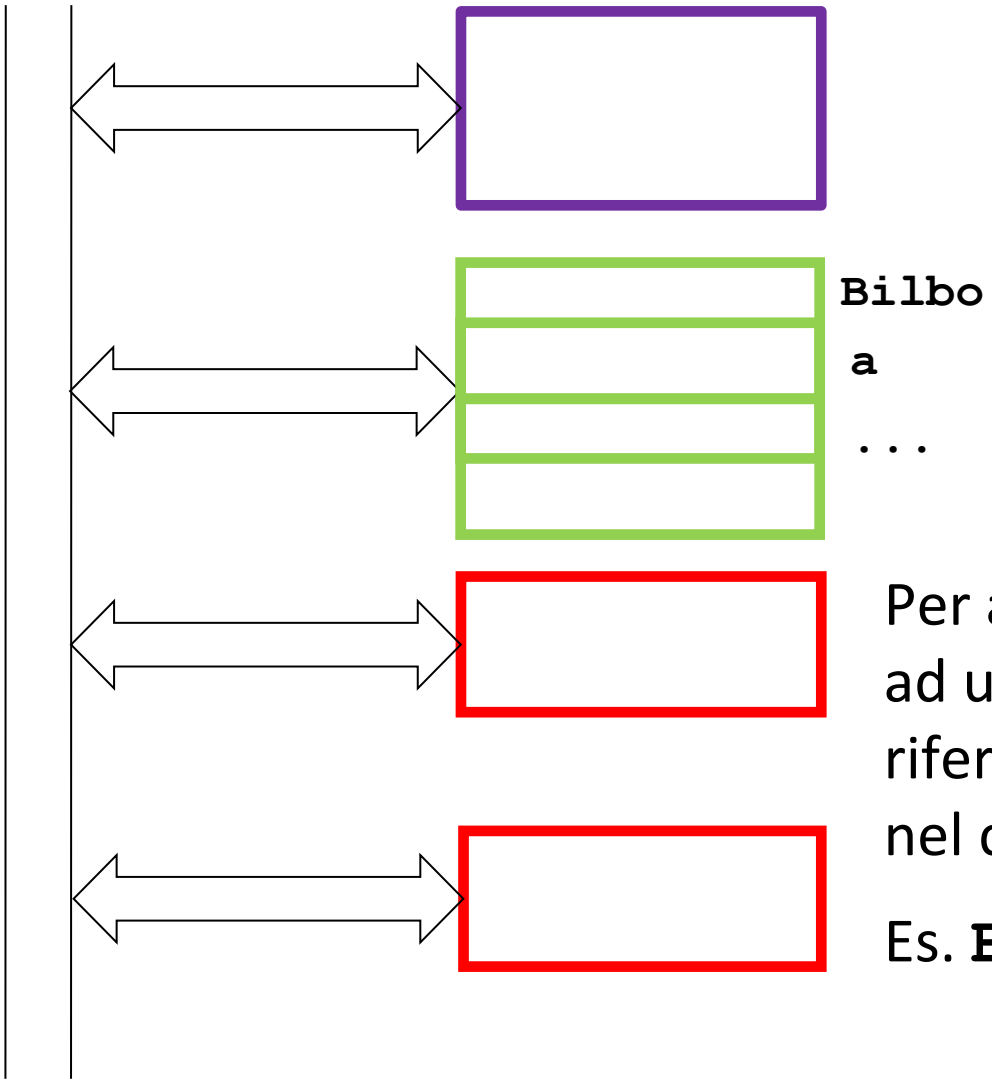
- Descrizione del linguaggio C mediante la macchina C che esegue i programmi codificati
- **Due sole periferiche:**
  - un'unica unità di ingresso, *Standard Input*
  - un'unica unità di uscita, *Standard Output*
- Standard Input, Standard Output e memoria **sono divisi in celle** elementari contenenti ciascuna un dato

Le Variabili

Linguaggio C: le Variabili

- **Variabile C: cella di memoria** nella macchina C
- Le variabili hanno un **nome**: un **identificatore simbolico** formato da successione di **lettere, cifre e carattere _** con al primo posto una lettera
  - Es. **`a, x, alfa, Bilbo, a1, Giuseppe, DopoDomani, velocita_massima`**
- Il linguaggio C è *case sensitive*, ovvero le maiuscole sono distinte dalle minuscole
  - Es. **`Alfa`**, **`alfa`** e **`ALPHA`** sono tre diversi identificatori
- Si chiamano **variabili** perché è possibile **cambiarne il contenuto** durante l'esecuzione del programma

Le Variabili: Identificatori Simbolici



Per accedere (in lettura o scrittura) ad una cella (in verde) mi basta far riferimento al nome della variabile nel codice

Es. **`Bilbo`**

Le Variabili: i Tipi

Tutte le variabili devono avere un **tipo dichiarato** che:

- caratterizza i valori scrivibili nella cella
- le operazioni sulla variabile
- la dimensione della cella in memoria (di questo non ce ne preoccupiamo)

Si dice anche che C è un programma **fortemente tipizzato**, ovvero devo sempre specificare il tipo di una variabile

Tipi Semplici Predefiniti (o Built-in)

- `char` permette di contenere un carattere della tabella ASCI, che corrisponde anche ad un intero in $[0,255]$
- `int` i numeri interi (anche negativi se non specificato diversamente), i valori massimi e minimi dipendono dalle dimensioni della parola della macchina
- `float` i numeri decimali a singola precisione
- `double` i numeri decimali a doppia precisione

Nel caso in cui una variabile avesse necessità di più di una cella di memoria, sarà C a occuparsene

Dichiarazione delle Variabili

- In C occorre **dichiarare ogni variabile**, specificandone il **nome ed il tipo**
- Sintassi per la dichiarazione

```
keywordTipo nomeVariabile;
```

Es. `int a;`

- È possibile dichiarare più variabili dello stesso tipo come `keywordTipo nomeVariabile1, nomeVariabile2;`

Es. `int a, b;`

Dichiarazione delle Variabili

- È possibile **dichiarare** ed **inizializzare** simultaneamente
  `keywordTipo nomeVariabile1 = valIniziale;`
  Es. `int a = 0, b = 8;`
- **Ogni variabile** deve essere **dichiarata** prima di essere utilizzata
- **Il valore** con cui vengono inizializzate le variabili **può essere modificato** mediante istruzioni di **assegnamento**

Le Istruzioni

Linguaggio C: le Istruzioni

- Le **istruzioni** sono **frasi eseguibili** del linguaggio,
- Ognuna **terminata dal simbolo ;** (punto e virgola)
- Le istruzioni, come le variabili hanno degli **identificatori**
- Tre tipi di istruzioni in C:
  - di assegnamento
  - di ingresso/uscita
  - composte

Le Istruzioni di Assegnamento

Sintassi:

```
nomeVariabile = espressione;
```

Assegna alla variabile **`nomeVariabile`** il valore di **`espressione`**

Dove **`nomeVariabile`** è l'identificativo di una variabile, mentre **`espressione`** è:

- un valore costante (e.g., 13, 'a', 2.7182)
- una variabile o una costante (ne considera il contenuto)
- combinazione di espressioni mediante **operatori** (es. aritmetici +, –, *, /) e parentesi

Esempi: `a = 7;` `k = 9.02;` `a = (3 - 214) * 2;` `a = b;` `a = a + 1;`

Le Istruzioni di Assegnamento

Sintassi:

```
nomeVariabile = espressione;
```

Assegna alla variabile **`nomeVariabile`** il valore di **`espressione`**

Ogni istruzione di assegnamento corrisponde a:

1. valutazione di **`espressione`** (leggendo il valore di eventuali variabili coinvolte)
2. memorizzazione del risultato nella cella identificata da **`nomeVariabile`**

NB: il simbolo '=' non indica uguaglianza/confronto: è l'operatore di assegnamento. Per confrontare due termini si usa l'operatore '=='

Le Istruzioni di Assegnamento: i Caratteri

- I caratteri alfanumerici vanno racchiusi tra apici singoli: ‘’
- Assegnamenti di un valore fissato ad una variabile **char** sono di questo tipo:

```
char a;
a = 'A';
a = 'z';
a = '1';
```

NB: l’ultima istruzione assegna alla variabile **a** il valore corrispondente al carattere `1`, che nella tabella ASCII corrisponde al numero 49

Operatori Aritmetici

- Vi sono gli operatori aritmetici **+, –, *, /** e le **parentesi tonde** per definire operazioni tra i valori delle variabili

NB: la divisione con l'operatore **/** assume diversi significati a seconda degli operandi:

- tra **`int`** calcola il quoziente troncato

```
int a,b;
float c;
c = a / b;
```

- tra **`float`** calcola il quoziente (con parte frazionaria)

```
int a,b;
float c;
c = (1.0 * a) / b;
```

```
float a, b, c;
c = a / b;
```

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”

Operatori Aritmetici

- Un nuovo operatore aritmetico: resto della divisione intera, o ***modulo*** `%`

  Es. `17%5` vale `2`, `15%5` vale `0`

- Segue che, con `a` e `b` interi: `a = (a/b)*b + a % b;`

Es.

```
int a = 11; int b = 4; int c;
```

`a = a + 1;` (viene scritto nella variabile `a` il valore `12`)

`c = a / b;` (viene scritto nella variabile `c` il valore `3`)

```
int a = 12; int b = 5; int c;
```

`c = a / b;` (viene scritto nella variabile `c` il valore `2`)

Operatori Aritmetici

- Un nuovo operatore aritmetico: resto della divisione intera, o ***modulo*** `%`

  Es. `17%5` vale `2`, `15%5` vale 0

- Segue che, con `a` e `b` interi: `a = (a/b)*b + a % b;`

Es.

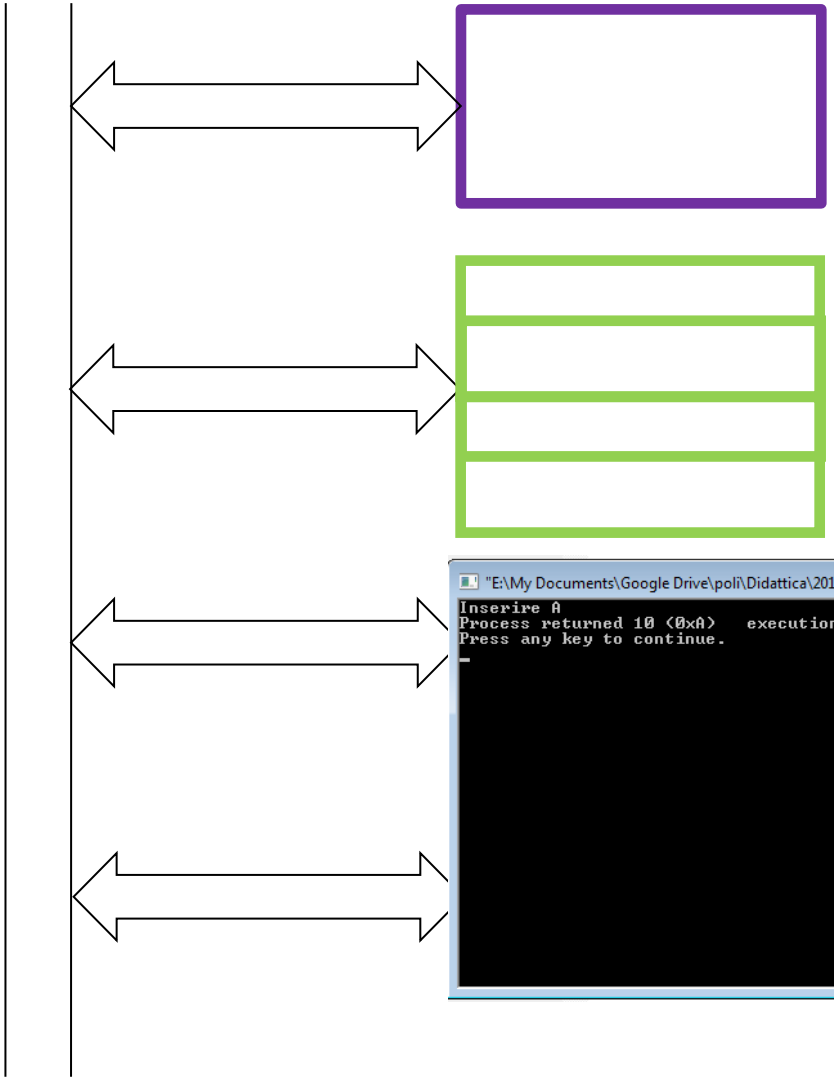
```
int a = 11; int b = 4; int c;
c = a % 2;
```
(viene scritto nella variabile `c` il valore `1`)

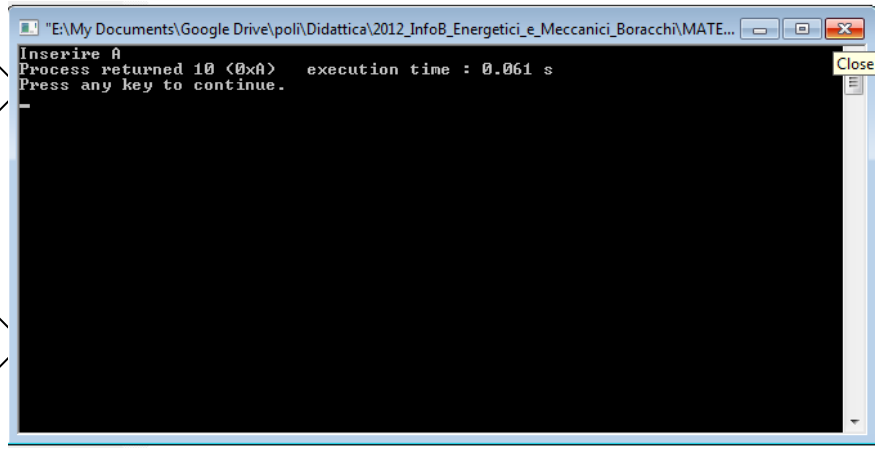
```
int a = 70; int b = 5; int c;
c = a % (b + 2);
```
(viene scritto in `c` il valore 0)

il valore di `b` non viene modificato, per modificare `b`:

```
b = b + 2; c = a % b;
```

Istruzioni Ingresso ed Uscita





- **`printf`** (uscita): scrittura su Standard Output
- **`scanf`** (ingresso): lettura da Standard Input e copia in una cella di memoria

Istruzioni Ingresso ed Uscita

- **Scrittura su schermo**: sintassi semplificata

```
printf (stringaControllo);
```

stringaControllo è una sequenza di caratteri racchiusa da apici doppi **(" ")**, i.e., una stringa
Cosa fa? Apre una finestra di dialogo e visualizza ***stringaControllo*** a schermo

- **Acquisizione da tastiera**: sintassi semplificata

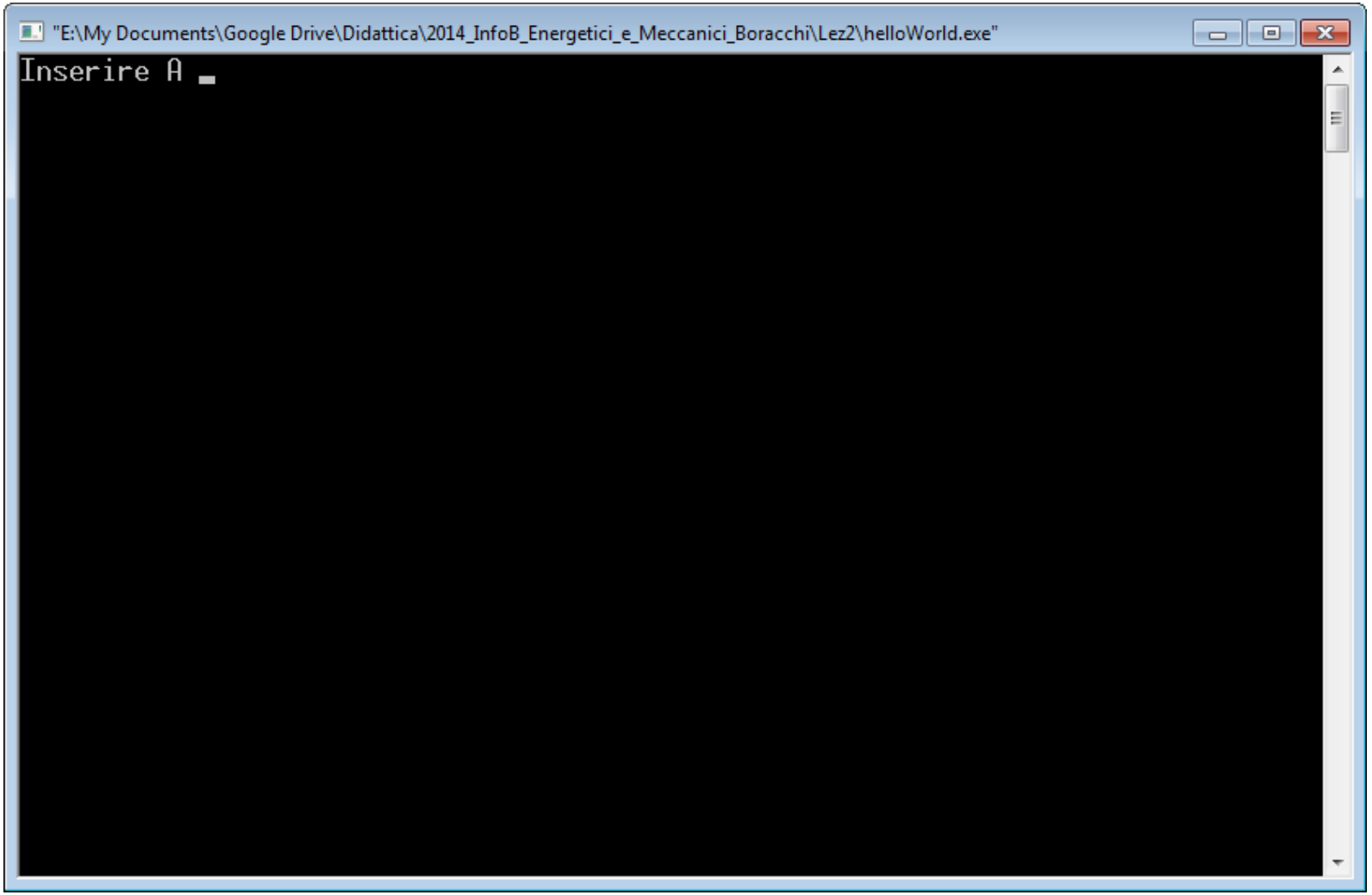
```
scanf("%d", &a);
```

a è una variabile intera dichiarata in precedenza

Cosa fa? Apre una finestra di dialogo e attende che l'utente digiti dei valori tastiera, il valore viene convertito in intero e copiato nella (cella di riferimento della) variabile **a**

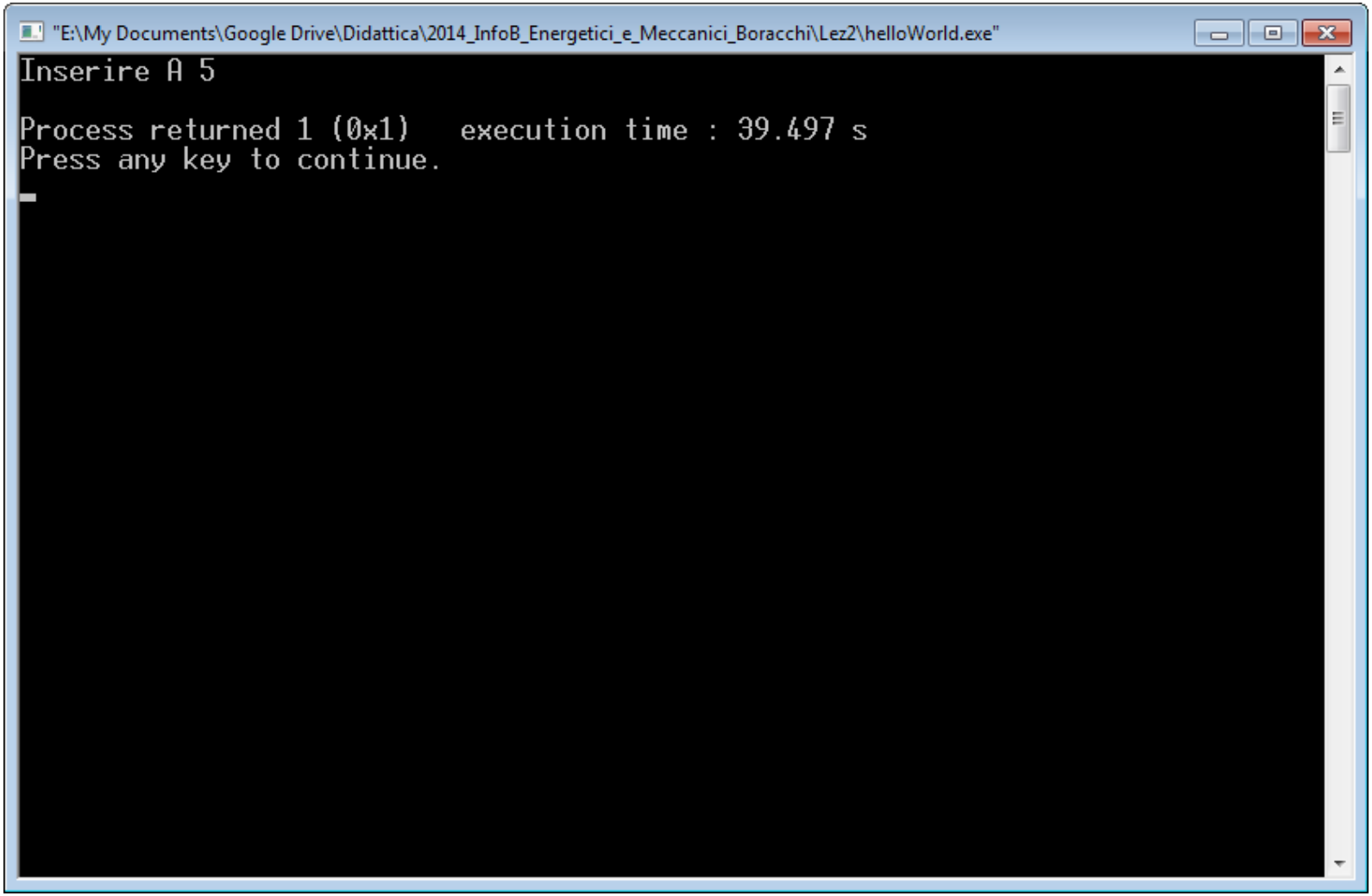
Esempio di Schermata di Dialogo per I/O

```
printf("Inserire A");
```



Esempio di Schermata di Dialogo per I/O

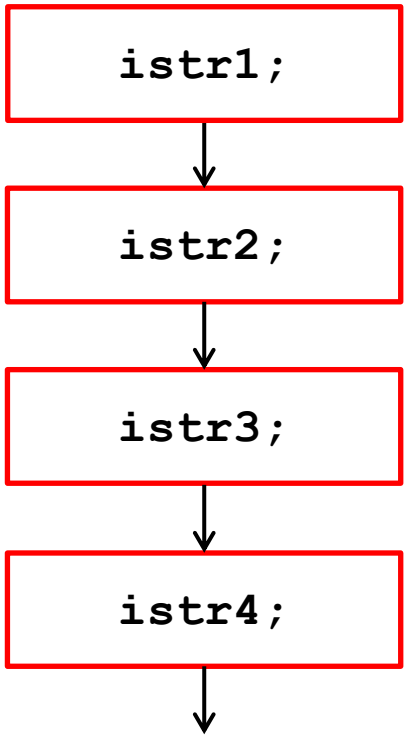
```
scanf("%d", &a);
```



La Sequenzialità

La Sequenza di Istruzioni

- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale,** dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*




```
istr1;
istr2;
istr3;
istr4;
...
```

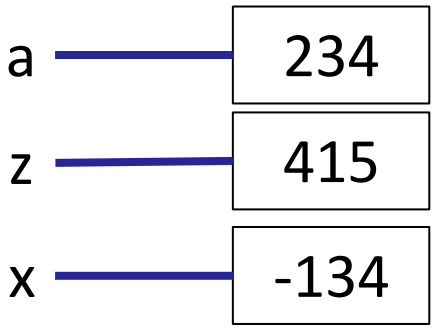
La Sequenza di Istruzioni

- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale,** dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

- *Es.*

```
int a,z,x;
a = 45;
z = 5;
x = (a - z) / 10;
```

- Stato della memoria



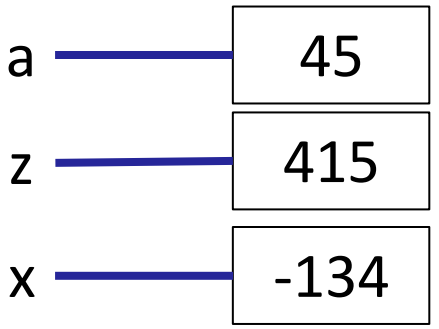
La Sequenza di Istruzioni

- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale,** dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

- *Es.*

```
int a,z,x;
a = 45;
z = 5;
x = (a - z) / 10;
```

- Stato della memoria



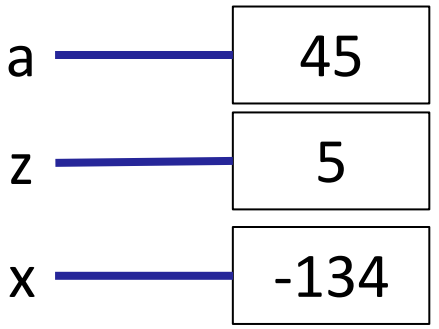
La Sequenza di Istruzioni

- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale,** dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

- *Es.*

```
int a,z,x;
a = 45;
z = 5;
x = (a - z) / 10;
```

- Stato della memoria



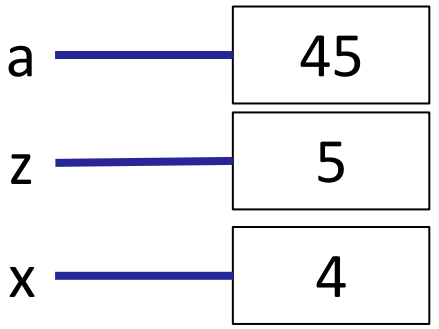
La Sequenza di Istruzioni

- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale,** dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

- *Es.*

```
int a,z,x;
a = 45;
z = 5;
x = (a - z) / 10;
```

- Stato della memoria



Il Mio Primo Programma

Struttura di un Programma C: Hello World

```
/* commenti
   commenti */
#include<stdio.h>
void main()
// punto di inizio
{
    printf("Hello world!");
}
```

Struttura di un Programma C: Hello World

```
/* commenti
   commenti */
#include<stdio.h>
void main()
// punto di inizio
{
    printf("Hello world!");
}
```

- I **commenti** sono ignorati dalla macchina e servono solo per facilitare la lettura e la scrittura del codice
  - `/* */` racchiude un commento su più righe
  - `//` precede un commento su una sola riga, fino al cambio di riga

Struttura di un Programma C: Hello World

```
/* commenti
   commenti */
#include<stdio.h>
void main()
// punto di inizio
{
    printf("Hello world!");
}
```

- Le prime istruzioni contengono le **direttive** per il compilatore
- `#include` serve per aggiungere istruzioni ai nostri programmi:

  `#include<nomeLibreria.h>`
  permette di utilizzare nel codice tutte le istruzioni (funzioni) presenti nella libreria `nomeLibreria.h`
- La libreria `stdio.h` (standard input/output) contiene funzioni per la gestione dell'Input e dell'Output, tra cui `printf` e `scanf`

Struttura di un Programma C: Hello World

```
/* commenti
   commenti */
#include<stdio.h>
void main()
// punto di inizio
{
    printf("Hello world!");
}
```

- Il punto da cui inizia l'esecuzione delle istruzioni è il **`main`**
- Il main rappresenta **l'intestazione** del programma
- **Ogni** programma C deve contenere il **`main`**
- Le istruzioni tra le parentesi graffe rappresentano il corpo del programma C
- La sintassi è:

```
int main(){…
        return 0;}
```

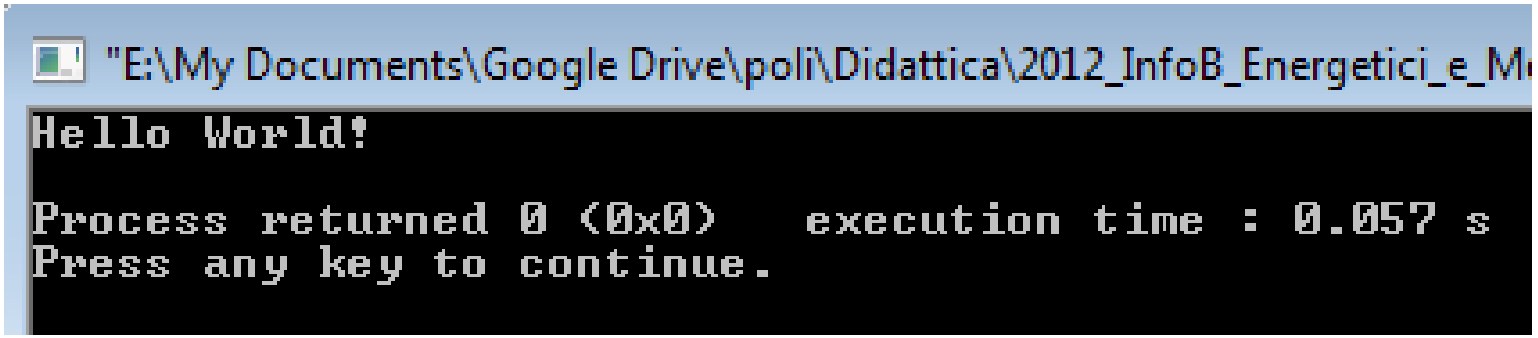
oppure

```
void main(){…}
```

Struttura di un Programma C: Hello World

```
/* commenti
   commenti */
#include<stdio.h>
void main()
// punto di inizio
{
   printf("Hello world!");
}
```

- Istruzione di stampa su standard output
- Il risultato dell'esecuzione di questo programma è quindi visualizzare a schermo la scritta:
  Hello world



Struttura di un programma C

```
/* eseguire la somma di due
numeri inseriti dall'utente*/
# include<stdio.h>
void main()
{
    int a, b, somma;
    printf("Inserire a:");
    scanf("%d" , &a);

    printf("Inserire b:");
    scanf("%d" , &b);

    somma = a + b;

    printf("\n %d + %d = %d",
            a, b, somma);
}
```

- Parte **dichiarativa** del programma
  - Dichiara le variabili e le costanti utilizzate dal programma
  - Facilita la diagnostica dei programmi (typos) e permette l'autocompletamento

Struttura di un programma C

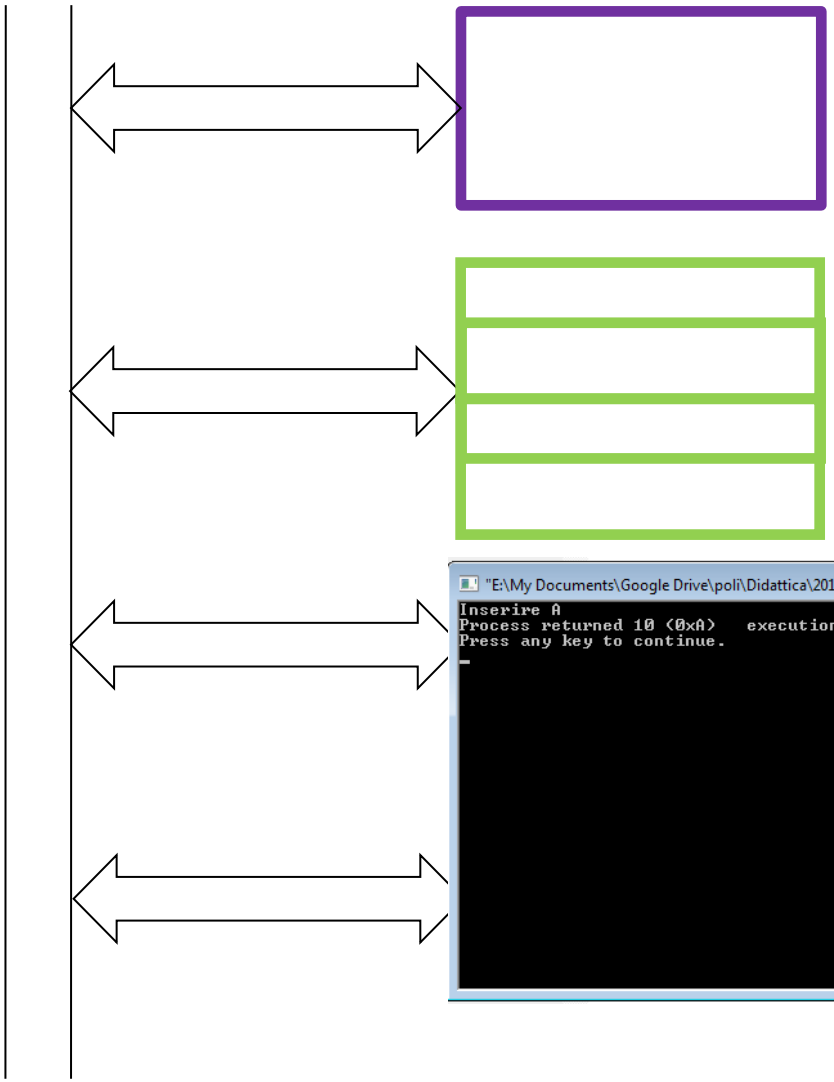
```
/* eseguire la somma di due
numeri inseriti dall'utente*/
# include<stdio.h>
void main()
{
    int a, b, somma;
    printf("Inserire a:");
    scanf("%d" , &a);

    printf("Inserire b:");
    scanf("%d" , &b);

    somma = a + b;

    printf("\n %d + %d = %d",
            a, b, somma);
}
```

- Parte **esecutiva** del programma, contiene le istruzioni del programma
- In questo caso:
  - I/O
  - Assegnamento

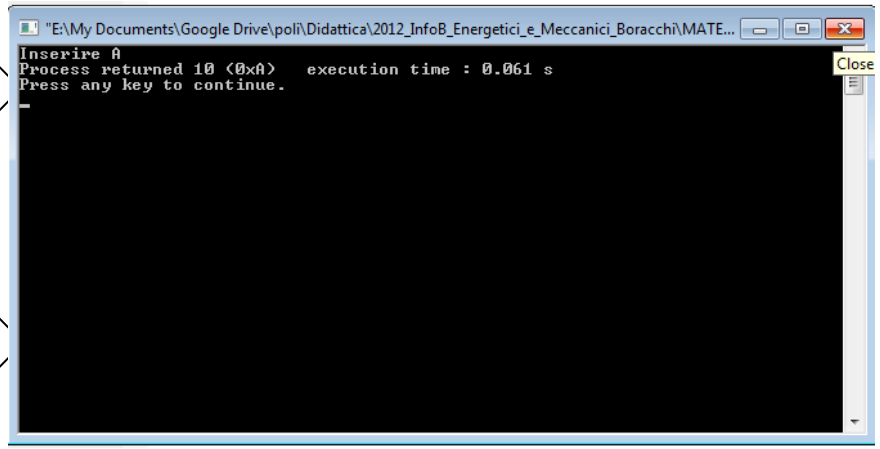
NB: l'incolonnamento dei programmi (tab e spazi) sono irrilevanti per il compilatore, ma facilitano la lettura del codice

Un po’ più di dettagli su I/O

Istruzioni Ingresso ed Uscita



- **`printf`** (uscita): scrittura su Standard Output
- **`scanf`** (ingresso): lettura da Standard Input e copia in una cella di memoria



Scrittura su Standard Output: `printf`

Esempio:

```
printf("\nInserire a:");
```

Sintassi:

```
printf (stringaControllo);
```

- ***stringaControllo*** può contenere:
  - **caratteri di stampa** (**normali** o **speciali**)

I caratteri nella ***stringaControllo*** vengono riportati a schermo

Scrittura su Standard Output: `printf`

Esempio:

```
printf("\n %d + %d = %d", a, b, a+b);
```

Sintassi:

```
printf (stringaControllo, elementiStampa) ;
```

- ***stringaControllo*** può contenere:
  - **caratteri di stampa** (**normali** o **speciali**)
  - **caratteri di conversione** (segnaposto, convertono valori di variabili in caratteri per la stampa)
- ***elementiStampa*** elenco di **variabili**, **espressioni composte,** o costanti separati da virgole

Ogni elemento di ***elementiStampa*** viene convertito in caratteri e associato ai caratteri di conversione in ***stringaControllo*** **nell'ordine con cui appare**

stringaControllo

- Alcuni **caratteri speciali per la stampa:**
  - `'\n'` manda a capo
  - `'\t'` spazia di un «tab»
- Alcuni **caratteri di conversione:**
  - `%d` intero decimale
  - `%f` numero reale
  - `%c` carattere
  - `%s` sequenza di caratteri (stringa)

NB: Non occorre specificare gli apici singoli per caratteri all'interno di stringhe

Scrittura su Standard Output: `printf`

Es:

```
int cat_dipend = 1;

float stip_medio = 35623.5;

printf ("Lo stipendio annuo dei dipendenti
di categoria %d è pari a $%f", cat_dipend,
stip_medio);
```

Scrittura su Standard Output: `printf`

Es:

```
int cat_dipend = 1;

float stip_medio = 35623.5;

printf ("Lo stipendio annuo dei dipendenti
di categoria %d è pari a $%f", cat_dipend,
stip_medio);
```

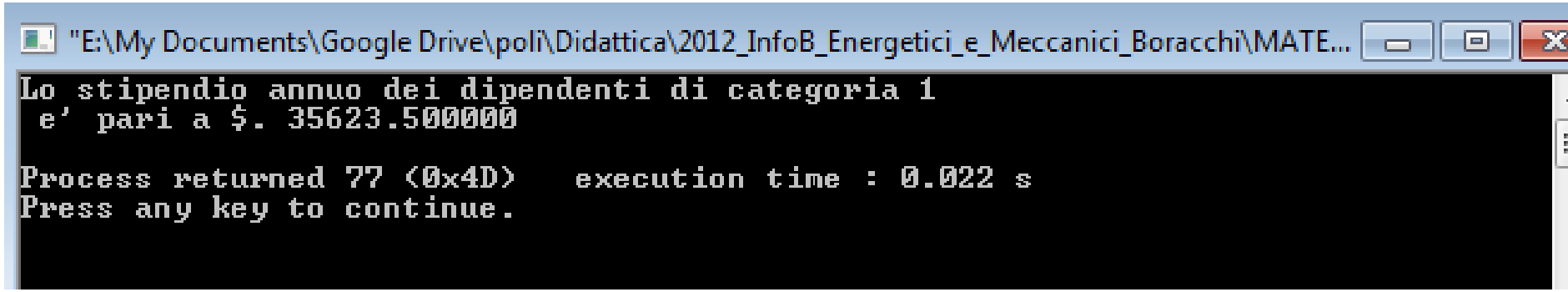
Nella stampa **`%d`** verrà sostituito dal valore di **`cat_dipend`** mentre **`%f`** verrà sostituito dal valore di **`stip_medio`**

L'abbinamento è dovuto **esclusivamente all'ordine** con cui appaiono i caratteri di conversione e le variabili (non al tipo)

Scrittura su Standard Output: `printf`

Es:

```
int cat_dipend = 1;

float stip_medio = 35623.5;

printf ("Lo stipendio annuo dei dipendenti
di categoria %d è pari a $%f", cat_dipend,
stip_medio);
```



Scrittura su Standard Output: `printf`

Es.

```
char iniz_nome = 'F';
char iniz_cognome = 'T';
printf("Questo programma è stato scritto da
\n%c%c\n\nBuon lavoro!\n", iniz_nome,
iniz_cognome);
```

Scrittura su Standard Output: `printf`

Es.

```
char iniz_nome = 'F';
char iniz_cognome = 'T';
printf("%s\n%c%c\n\n%s\n", "Questo programma è stato scritto da", iniz_nome, iniz_cognome, "Buon lavoro!");
```

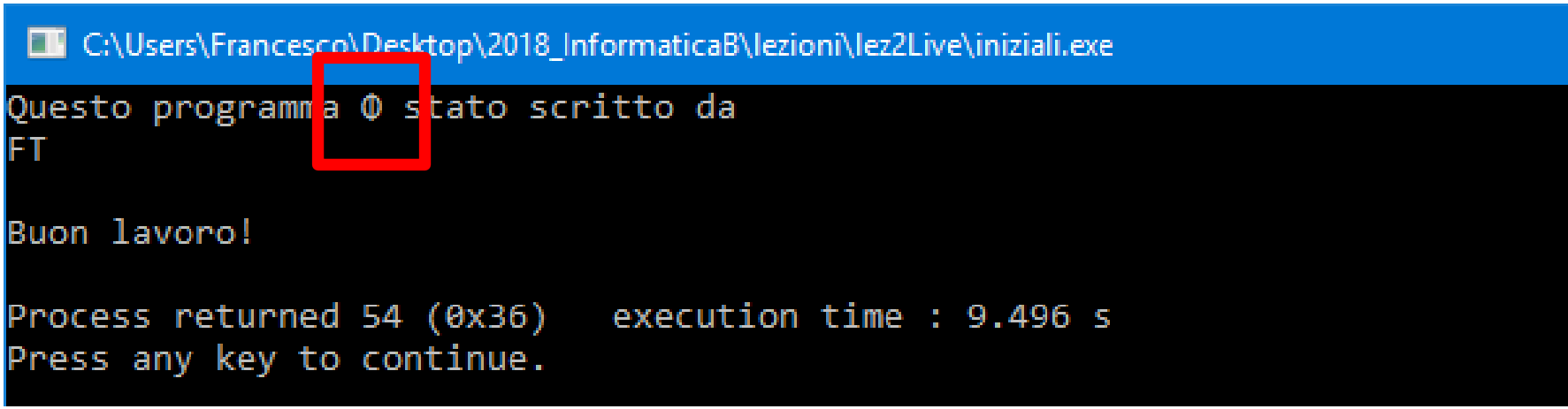
- È possibile specificare anche le stringhe (sequenze di caratteri) al di fuori della ***stringaControllo***, purchè a queste si faccia riferimento con un carattere di conversione `%s`

NB: non esiste un tipo di variabile built-in per contenere una stringa

Scrittura su Standard Output: `printf`

Es.

```
char iniz_nome = 'F';
char iniz_cognome = 'T';
printf("Questo programma è stato scritto da
\n%c%c\n\nBuon lavoro!\n", iniz_nome,
iniz_cognome);
```



Lettura da Standard Input: `scanf`

- Esempio:

```
scanf("%d", &b);
```

- Sintassi:

`scanf(`*`stringaControllo`*`, indirizzoVariabile`*`)`

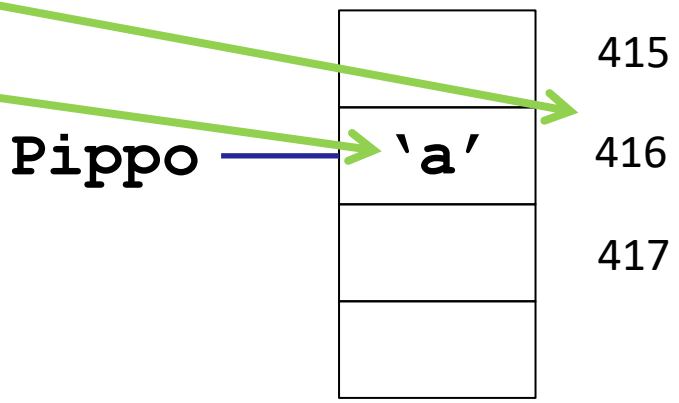
  - ***`stringaControllo:`*** una **stringa di caratteri** di conversione che specifica il tipo del dato inserito da tastiera
  - **`indirizzoVariabile`: indirizzo** in memoria di una cella associata ad una variabile inizializzata nel programma

- Acquisice dei valori da standard input, li converte nel tipo specificato da ***`stringaControllo`***, li copia nella variabile all'indirizzo **`indirizzoVariabile`**

Indirizzo di una Variabile

- L'indirizzo di una variabile si ottiene anteponendo l'operatore **&** al nome della variabile

```
char Pippo;

Pippo = 'a';

&Pippo

Pippo
```



Lettura da Standard Input: scanf

Esempi di acquisizione di diversi tipi:

- ```
  int x;
  scanf("%d", &x);
  ```
- ```
  float x;
  scanf("%f", &x);
  ```
- ```
  double x;
  scanf("%f", &x);
  ```

Acquisizioni multiple sono possibili, ma sconsigliate

- ```
  int x,y;
  float z;
  scanf("%d%d%f", &x,&y,&z);
  ```

Qualche Programma più Elaborato

Esercizio: Conversione in Celsius

- Scrivere un programma che prende in ingresso una temperatura in Fahrenheit e la trasforma in Celsius
- La formula di conversione é $C = 5/9 * (F - 32)$

```
// conversione da gradi Fahrenheit
a Celsius
#include <stdio.h>
void main(){
      int    Ftemp;
      float Ctemp;
      printf("Inserire la
temperatura in Fahrenheit da
convertire in Celsius\n");
      scanf("%d", &Ftemp);
      Ctemp = (5.0/9)*(Ftemp - 32);
      printf("in Celsius %f" ,
Ctemp);
}
```

TODO

- Estendere il programma precedente per prendere in ingresso un prezzo totale con anche i centesimi e dire:
  - Quante banconote da 50, 10 e 5
  - Quante monete da 2 e 1 Euro
  - Quanto (in totale) in monete minori

Esercizio: Swap di Due Caratteri

```
# include<stdio.h>
void main()
{
 char a,b,c;
 printf("\nInserire il carat. A = ");
 scanf("%c" , &a);
 fflush(stdin);
 printf("\nInserire il carat. B = ");
 scanf("%c" , &b);

 c = a; // salvo in c il valore di a
 a = b;
 b = c;
 printf("\nA = %c", a);
 printf("\nB = %c", b);
}
```

- Scrivere un programma che richiede due caratteri che vengono salvati in opportune variabili
- Il programma poi scambia i contenuti delle variabili e ne stampa i valori

Esercizio: Swap di Due Caratteri

```
# include<stdio.h>
void main()
{
 char a,b,c;
 printf("\nInserire il carat. A = ");
 scanf("%c" , &a);
 fflush(stdin);
 printf("\nInserire il carat. B = ");
 scanf("%c" , &b);

 c = a; // salvo in c il valore di a
 a = b;
 b = c;
 printf("\nA = %c", a);
 printf("\nB = %c", b);
}
```

`fflush(stdin);`

Serve per pulire il buffer di ingresso, lo standard input

All'inserimento del primo carattere premo anche un invio – per confermare l'inseriment e questo invio rimane nel buffer di ingresso e viene acquisito da **`scanf("%c", &b);`**

Esercizio: Swap di Due Variabili

1. Prendi un terzo bicchiere C
2. Rovescia il contenuto del bicchiere A nel bicchiere C
3. Rovescia il contenuto di B in A
4. Rovescia il contenuto di C in B

NB: Esise un algoritmo per scambiare i valori di due variabili A e B per cui non occorre il bicchiere C