Introduzione al Linguaggio C

Informatica A AA 2023 / 2024

Giacomo Boracchi

giacomo.boracchi@polimi.it

22 Settembre 2023

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.

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.

Programmazione a Basso / Alto Livello

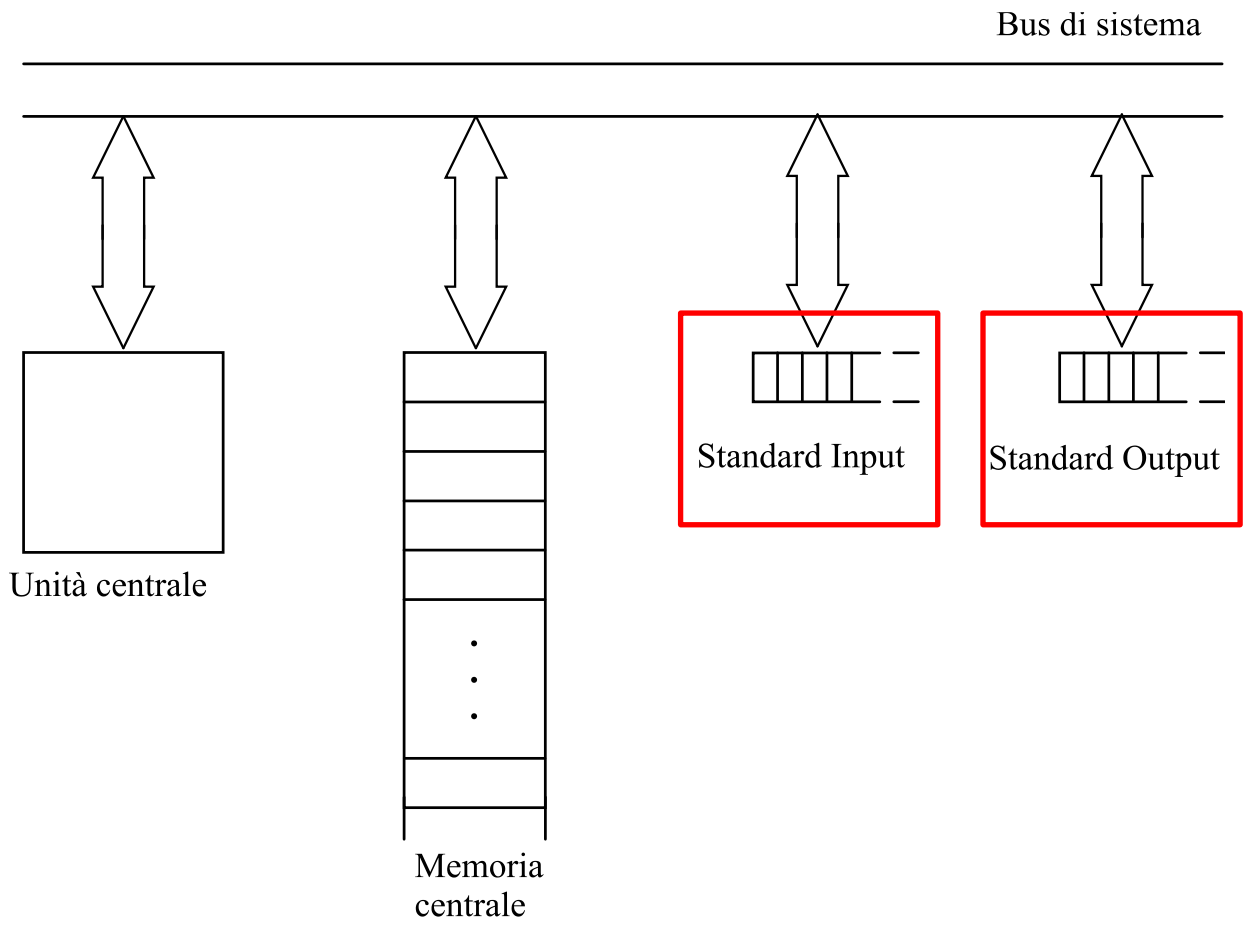
- **Linguaggio macchina**: poche istruzioni, difficile codificare algoritmi e interpretare il codice
  - Linguaggio preciso.
  - Completo controllo delle risorse.
- **Linguaggi di alto livello**: linguaggi più comprensibili per l'uomo.
  - Linguaggio preciso e sintetico
  - Riferimenti simbolici
  - 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

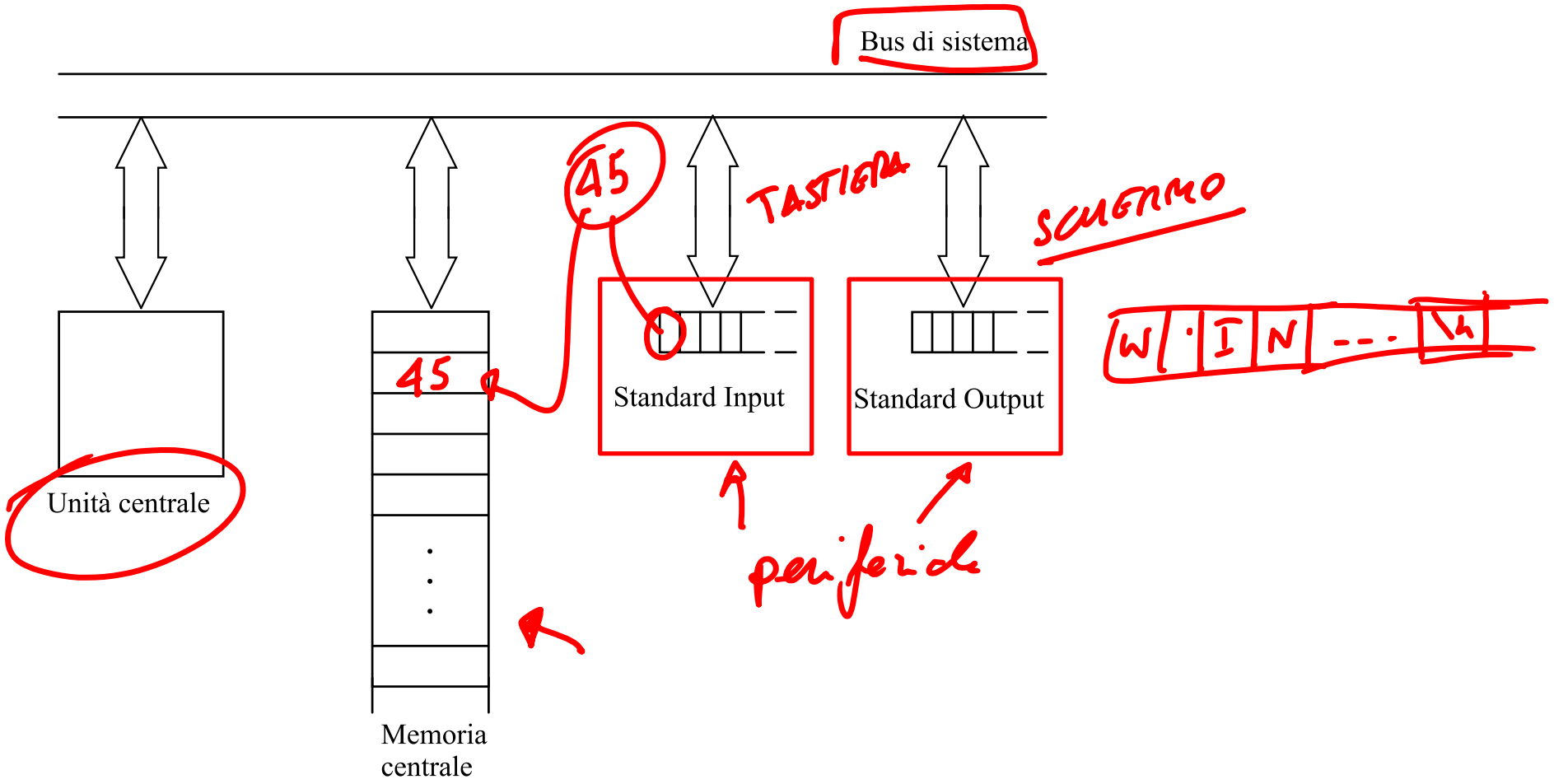
Il Linguaggio C, qualche informazione storica

- 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 linguaggio C per diverse architetture
- Nel 1983, l’ANSI (American National Standards Institute) lavora ad una versione standard del C, e anche recentemente con l’ANSI99
- Ad oggi, gran parte dei sistemi operativi è ancora scritta in C (o C++)

Descrizione del linguaggio C mediante la macchina C che esegue i programmi codificati.



Descrizione del linguaggio C mediante la macchina C che esegue i programmi codificati.



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

Il Mio Primo Programma

Struttura di un programma C

```
/* commenti
   commenti */
#include<stdio.h>
int main()
// punto di inizio
{
    printf("Hello world!");
    return 0;
}
```

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

```
/* commenti
   commenti */
#include<stdio.h>
int main()
// punto di inizio
{
    printf("Hello world!");
    return 0;
}
```

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

  `#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

```
/* commenti
   commenti */
#include<stdio.h>
int main()
// punto di inizio
{
    printf("Hello world!");
    return 0;
}
```

- Il punto da cui inizia l'esecuzione delle istruzioni è il **main.**
- Il main rappresenta **il punto di inizio** del programma ed **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;}
```

Ma è possibile anche

```
void main(){…}
```

Struttura di un programma C

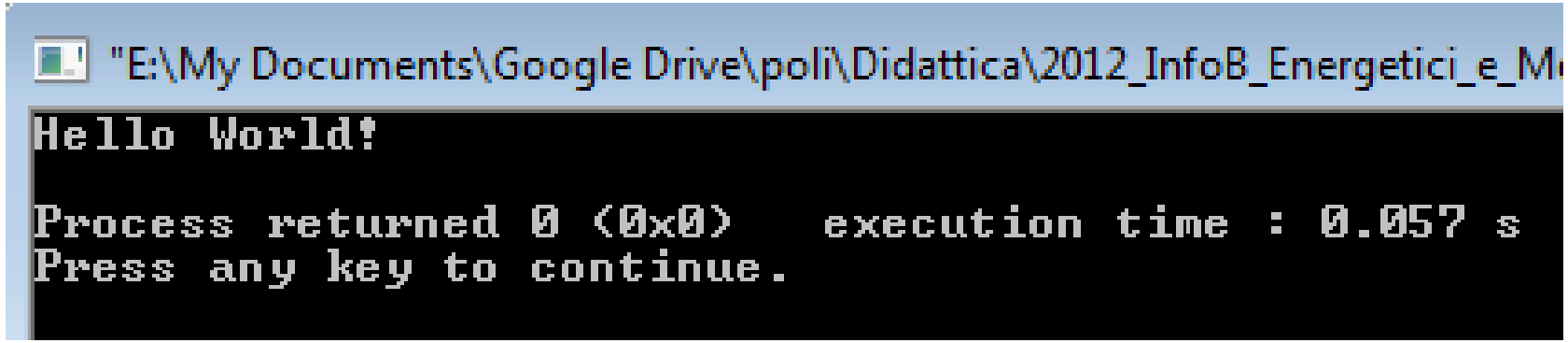
```
/* commenti
   commenti */
#include<stdio.h>
int main()
// punto di inizio
{
   printf("Hello world!");
   return 0;
}
```

- 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

```
/* commenti
   commenti */
#include<stdio.h>
int main()
// punto di inizio
{
   printf("Hello world!");
   return 0;
}
```

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



Osservazioni

- **`int main() { …` *corpo* `… }`**
  - I programmi C contengono la definizione di una o più funzioni, una delle quali deve essere il **`main()`**
    - In questo caso c'è la definizione della sola funzione main()
  - Le parentesi tonde ( ) indicano che è una **funzione**
  - Le funzioni tipicamente "restituiscono" un valore
    - **`int`** significa che il **`main`** restituisce un valore intero
  - Il ***corpo*** di ogni funzione è incluso in un ***blocco***, racchiuso in parentesi graffe { }
    - Il corpo costituisce la definizione della funzione. Raccoglie le istruzioni che specificano il "comportamento" della funzione.

Osservazioni

```
printf ("Hello World!\n");
```

- È una ***istruzione*** di stampa
  - L'intera riga si chiama istruzione (o *statement*)
  - Visualizza una sequenza di caratteri indicata tra le doppie virgolette `"..."`
- L'istruzione richiede l'esecuzione di una funzione (la funzione printf)
  - Si dice che l'istruzione costituisce una *chiamata* alla funzione
    - La funzione è definita altrove (in una libreria standard)
    - Nel nostro programma c' è solo la "chiamata"
  - Le parentesi tonde raccolgono I parametri passati alla funzione
    - f(x) ... printf ("...stringa...") Qui il parametro è una stringa
- Il carattere `'\'` (*backslash*) si premette ai cosiddetti "caratteri di escape":
  - il **carattere `'\n'`** indica che **`printf()`** deve fare qualcosa di speciale
  - è il carattere di escape che indica "new line" (vai a capo)

Osservazioni

`return 0;`

- È un modo di terminare una funzione
- Indica il valore "restituito" come effetto dell'esecuzione
- È compatibile con la dichiarazione **`int main()`**
- **`return 0`**, in questo caso significa che il programma è terminato senza anomalie
  - È una semplice convenzione: **`0`** indica la fine di una esecuzione corretta, ad altri valori si associano interpretazioni legate al tipo di errore che si è verificato
  - È una convenzione tipica del **`main`**. Per le altre funzioni normalmente si usano convenzioni e interpretazioni legate al significato della funzione

Linker (o collegatore)

- Quando si chiama una funzione (**`printf`** nell' esempio), il collegatore la cerca nelle librerie
  - nella libreria standard, ed eventualmente in quelle indicate da una apposita direttiva al preprocessore
- La funzione è copiata dalla libreria e inserita nel programma oggetto
- Se c'è un errore nel nome della funzione il collegatore lo rileva (non riesce a trovarla)

Le Variabili

.. I foglietti che abbiamo usato per scrivere algoritmi...

Le variabili

`int integer1, integer2, sum,` sono delle **variabili**

Variabili: riferimenti simbolici a porzioni di memoria dove sono **memorizzati dati** che possono essere manipolati dal programma

- `int` : tipo delle varibiali, specifica che le variabili conterranno numeri interi
- `integer1`, `integer2`, `sum` - nomi di variabili

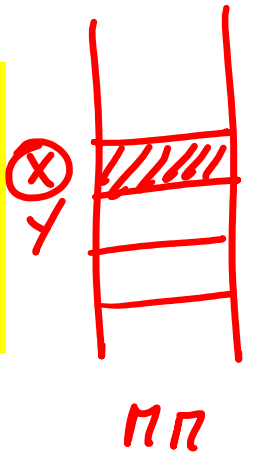
Le variabili

`int integer1, integer2, sum,` sono delle **variabili**

Variabili: riferimenti simbolici a porzioni di memoria dove sono **memorizzati dati** che possono essere manipolati dal programma

- `int` : tipo delle varibiali, specifica che le variabili conterranno numeri interi
- **`integer1`**, **`integer2`**, **`sum`** - nomi di variabili

Le variabili si chiamano variabili perché il loro valore può cambiare durante l'esecuzione del programma



`int integer1, integer2, sum,` sono delle **variabili**

Variabili: riferimenti simbolici a porzioni di memoria dove sono **memorizzati dati** che possono essere manipolati dal programma

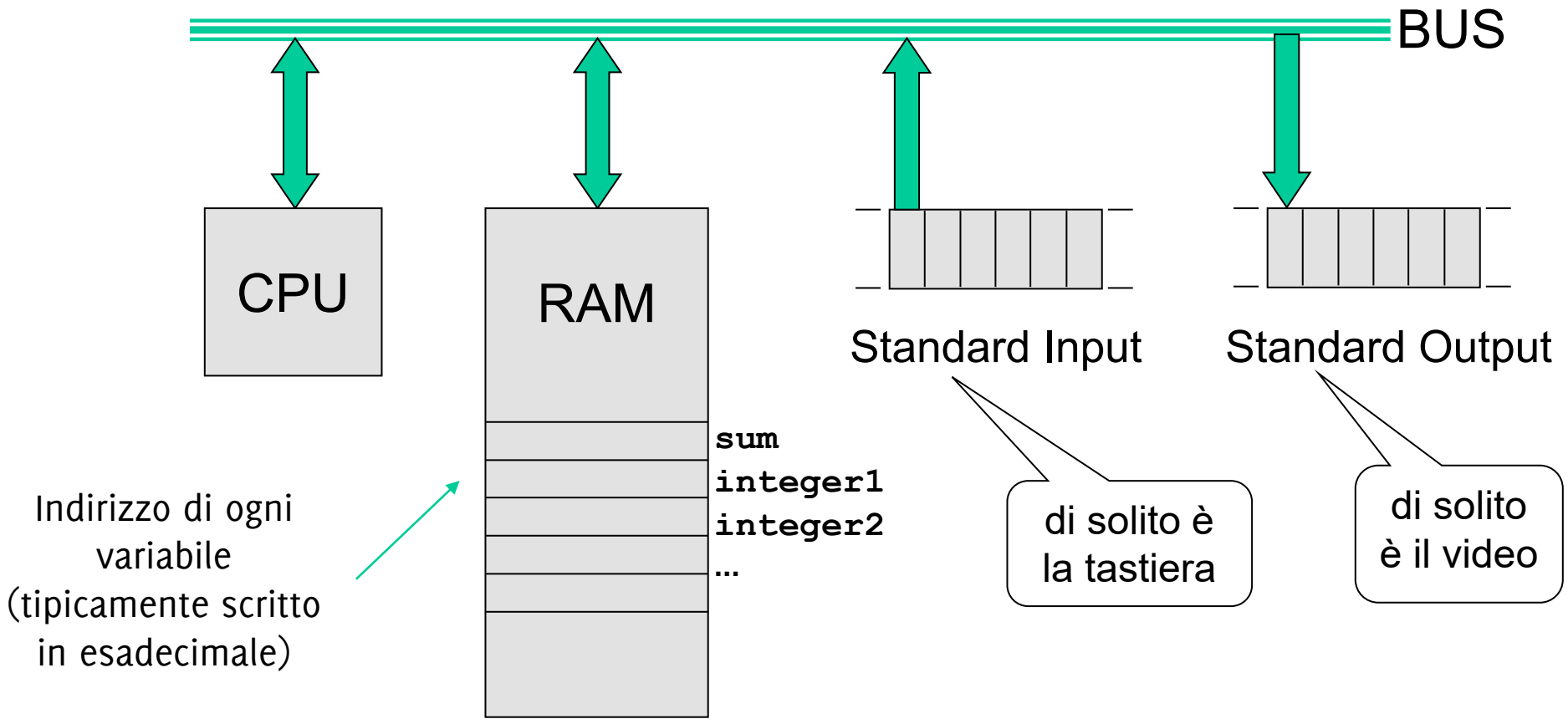
- `int` : tipo delle varibiali, specifica che le variabili conterranno numeri interi
- `integer1`, `integer2`, `sum` - nomi di variabili

Per capirci di più dobbiamo conoscere il modello di esecutore per i programmi C

La macchina astratta C

Algoritmi e programmi sono definiti in funzione del loro esecutore

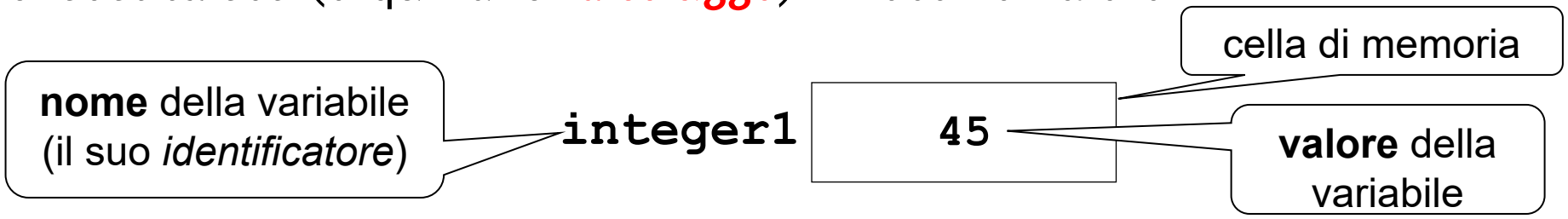
L'esecutore dei programmi C è una macchina astratta



Le variabili in memoria centrale

Variabili: riferimenti simbolici che corrispondono a *locazioni* di memoria

- Ogni variabile ha
  - un ***nome*** *(identificatore)*
  - un ***tipo*** (insieme dei *valori* e delle *operazioni* ammissibili)
  - una ***dimensione***
  - un ***indirizzo*** *(individua la cella [o le celle] di memoria)*
  - un ***valore***
- **La lettura *non modifica*** i valori delle variabili
- **Inserendo un nuovo valore** in una variabile (per assegnamento o con una **scanf**), **si *sostituisce*** (e quindi si **distrugge**) il vecchio valore



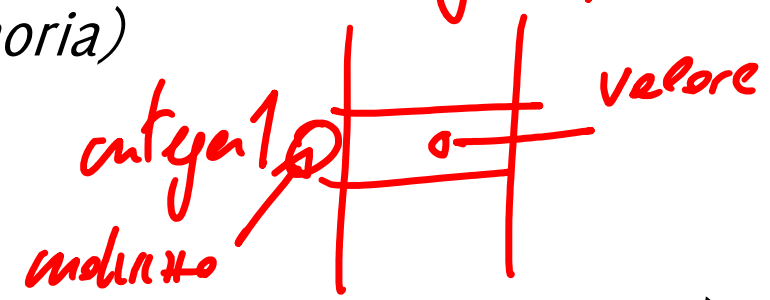
Le variabili in memoria centrale

Variabili: riferimenti simbolici che corrispondono a *locazioni* di memoria

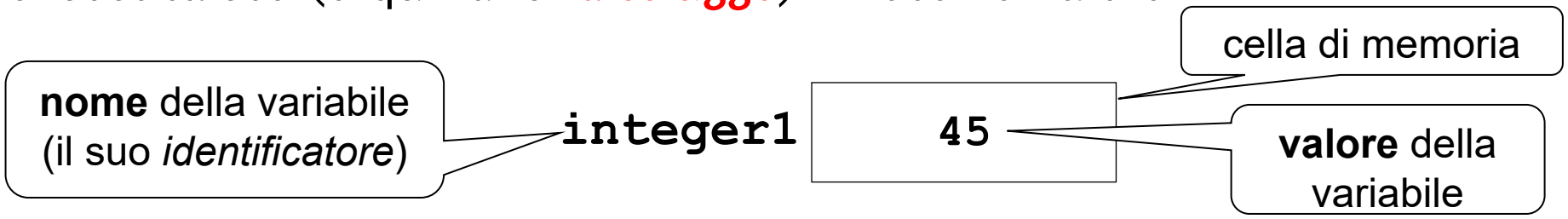
- Ogni variabile ha
  - un ***nome*** *(identificatore)*
  - un ***tipo*** (insieme dei *valori* e delle *operazioni* ammissibili)
  - una ***dimensione***
  - un ***indirizzo*** *(individua la cella [o le celle] di memoria)*
  - un ***valore***

tipo nomeVariabile

int integer1;



- **La lettura *non modifica*** i valori delle variabili
- **Inserendo un nuovo valore** in una variabile (per assegnamento o con una **scanf**), **si *sostituisce*** (e quindi si **distrugge**) il vecchio valore



Le variabili

Rappresentano i *dati su cui lavora il programma*

Sono denotate mediante identificatori:

```
a, B, Pluto, somma_1, …
```

Variabili diverse devono avere **identificatori diversi**

Alla stessa variabile ci si riferisce sempre con lo **stesso identificatore**

Durante l'esecuzione del programma le variabili hanno **sempre un valore** ben definito

- Può essere significativo o non significativo, ma c'è
- Perciò le variabili **devono essere sempre inizializzate** in modo opportuno [per evitare errori e sorprese]

Le variabili

Rappresentano i *dati su cui lavora il programma*

Sono denotate mediante identificatori:

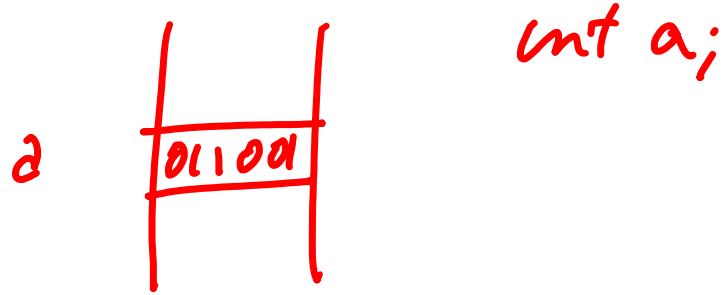
```
a, B, Pluto, somma_1, …
```

Variabili diverse devono avere **identificatori diversi**

Alla stessa variabile ci si riferisce sempre con lo **stesso identificatore**

Durante l'esecuzione del programma le variabili hanno **sempre un valore** ben definito

- Può essere significativo o non significativo, ma c'è
- Perciò le variabili **devono essere sempre inizializzate** in modo opportuno [per evitare errori e sorprese]



Variabile ⇔ cella di memoria nella macchina C

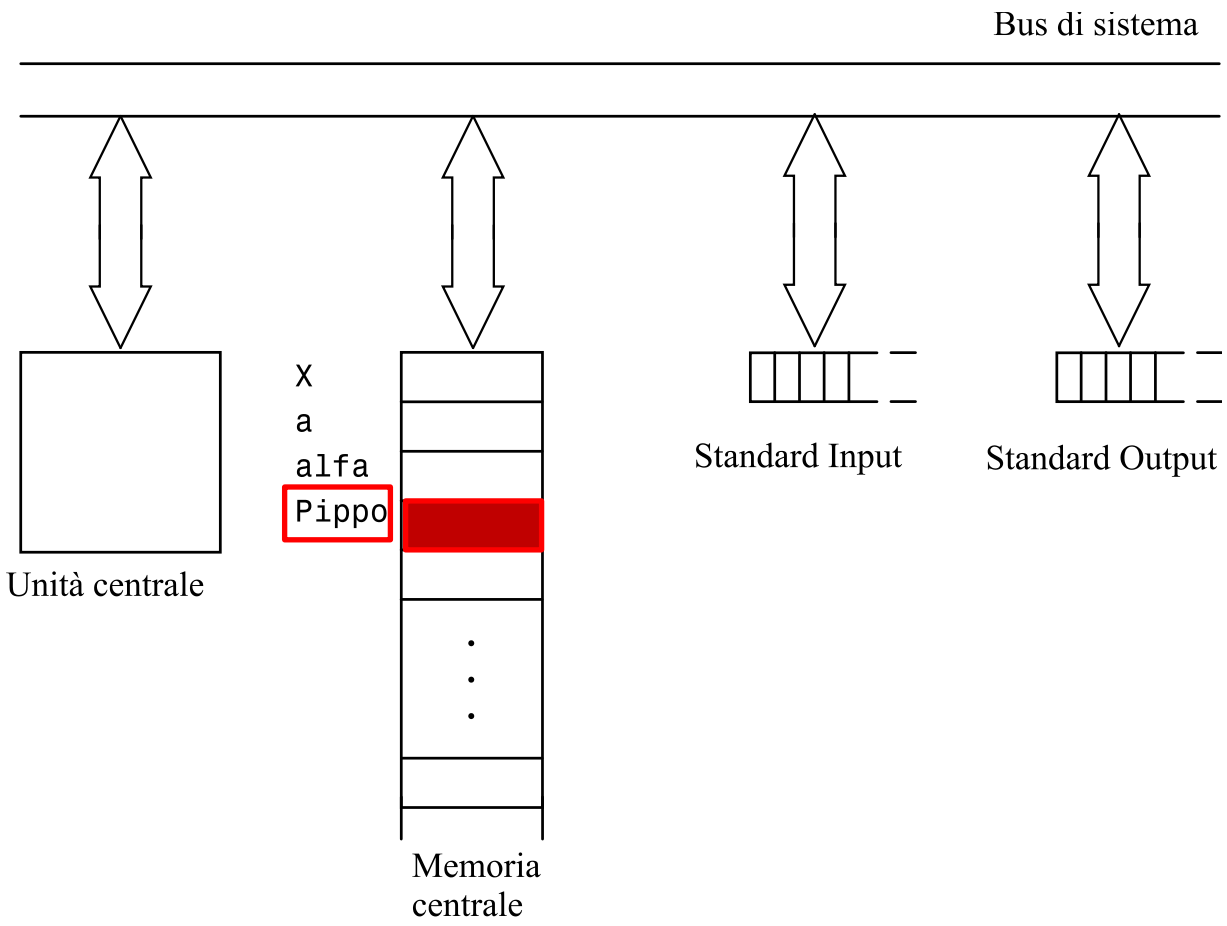
Le variabili hanno un **nome**: un **identificatore simbolico** formato da successione di **lettere, cifre e carattere _** con al primo posto una lettera.

- Es. di identificatori in C: **`a`, `x`, `alfa`, `pippo`, `a1`, `Giuseppe`, `DopoDomani`, `velocita_massima`**
- **NB**: maiuscole distinte dalle minuscole (**`Alfa`**, **`alfa`** e **`ALPHA`** sono tre diversi identificatori).

Si chiamano **variabili** perché è possibile **cambiarne il contenuto**

Le Variabili, identificatori simbolici

Per accedere (in lettura o scrittura) alla cella in rosso mi basta far riferimento alla variabile **`Pippo`** nel codice.



le Variabili (cnt)

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)

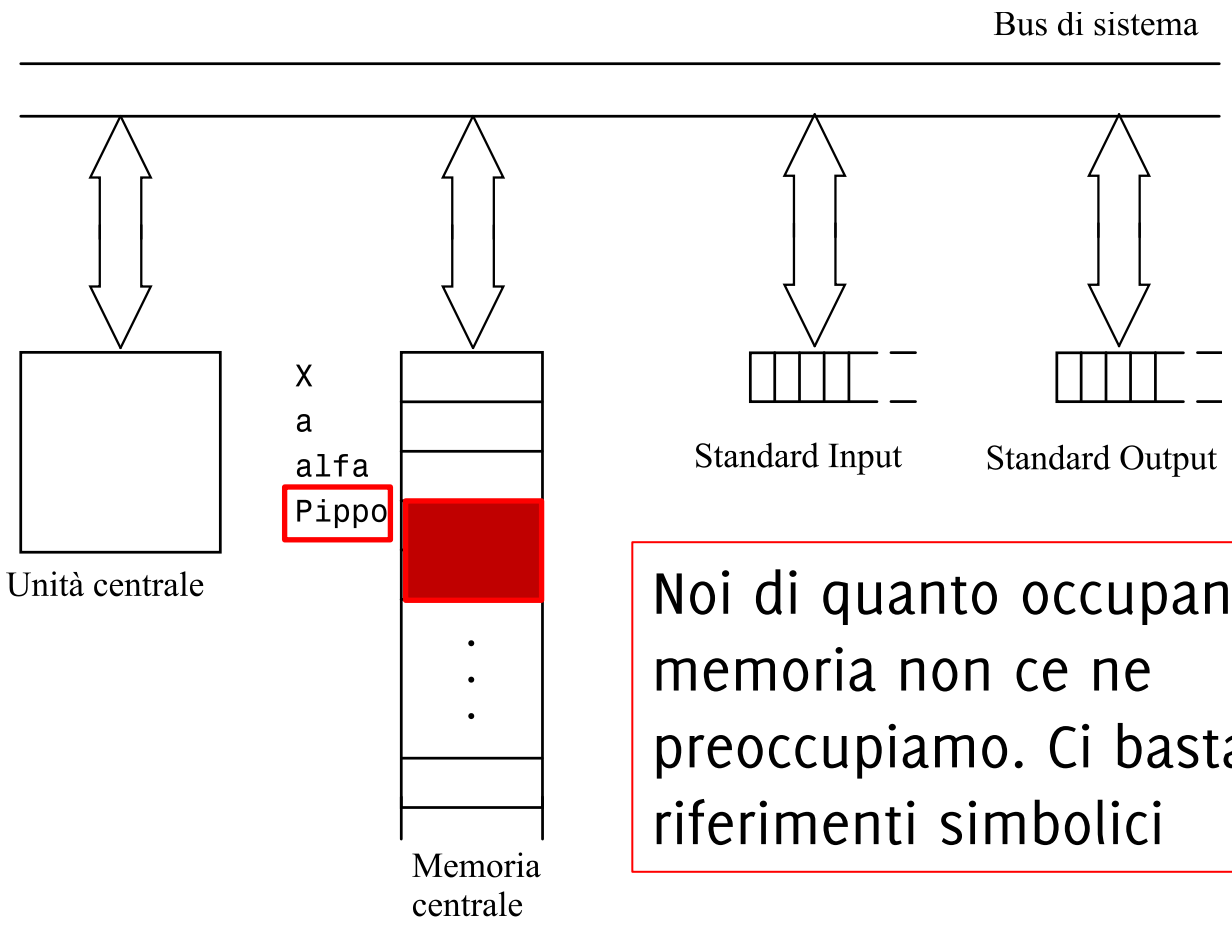
Tipi semplici predefiniti (built in)

- `char` permette di contenere un carattere della tabella ASCI che corrisponde ad un intero [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

Queste parole sono **keywords,** non posso usarle per altri scopi

Le Variabili, identificatori simbolici

A seconda del tipo è possibile che più celle facciano riferimento alla stessa variabile (**`Pippo`**)



Noi di quanto occupano in memoria non ce ne preoccupiamo. Ci basta usare i riferimenti simbolici

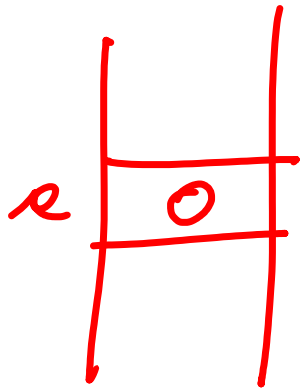
Dichiarazione delle Variabili

- 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 (cnt.)

- 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 (per questo si chiamano variabili!).

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`**.

Sintassi:

```
nomeVariabile = espressione;
```

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

Non è un'operazione di confronto!

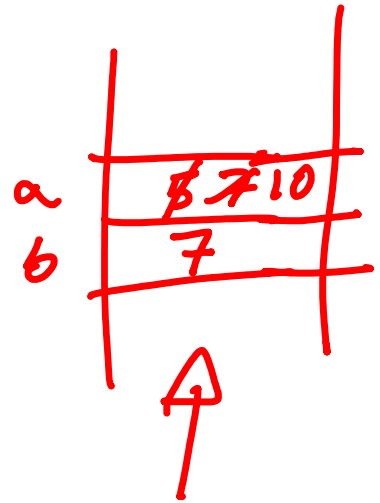
Sintassi:

= è costruzione di ASSEGNAMENTO NON CONFRONTO

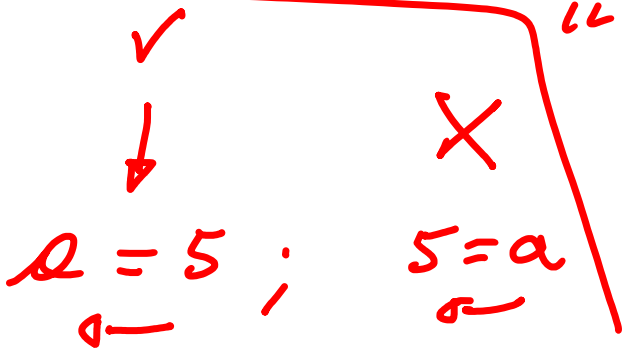
```
nomeVariabile = espressione;
```

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

```
int a,b;
a = 5;
b = a+2;
a = b;
a = a+3;
```



1) RISOLVI ESPRESSIONE
2) SCRIVI IN a IL VALORE DI ESPRESSIONE



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; // copia valore di b in a, DIVERSO da b = a;

a = a + 1; // questa non è un'equazione priva di soluzione
```

Le Istruzioni di Assegnamento (cnt.)

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: simbolo '=' non indica uguaglianza/confronto: è l'operatore di assegnamento.

Per confrontare due termini si usa l'operatore '=='

Operazioni sulle variabili: assegnamento

= (operatore di assegnamento)

- Assegna un valore a una variabile
- È un operatore *binario* (cioè che ha due operandi):

```
sum = integer1 + integer2;
```

sum *assume il valore* **integer1 + integer2**

```
int main(){
  int integer1, integer2, sum;

  printf("Enter first integer\n");
  scanf("%d", &integer1 );
  printf("Enter second integer\n");
  scanf("%d", &integer2);
  sum = integer1 + integer2;
  printf("Sum is %d\n\n", sum );

  return 0;
}
```

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';
```

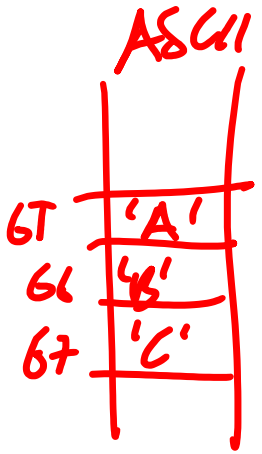
- **N.B: l’ultima istruzione assegna alla variabile a il valore corrispondente al carattere 1 che nella tabella ASCII corrisponde al numero 49**

Le Istruzioni di Assegnamento: i caratteri

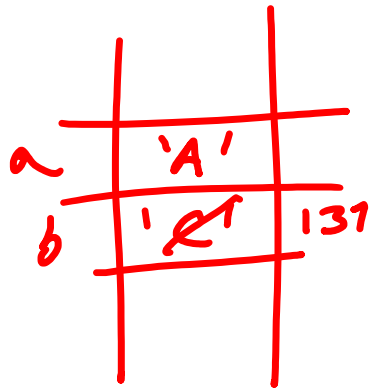
- I caratteri alfanumerici vanno racchiusi tra apici singoli: ‘ ’
- Assegnamenti di un valore fissato ad una variabile **char** sono di questo tipo

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

- **N.B: l’ultima istruzione assegna alla variabile a il valore corrispondente al carattere 1 che nella tabella ASCII corrisponde al numero 49**




```
char a, b;
a = 'A';
b = a + 2;
b = 'A' + 'B';
```



Operatori Aritmetici

Operatori Aritmetici

Vi sono i soliti 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

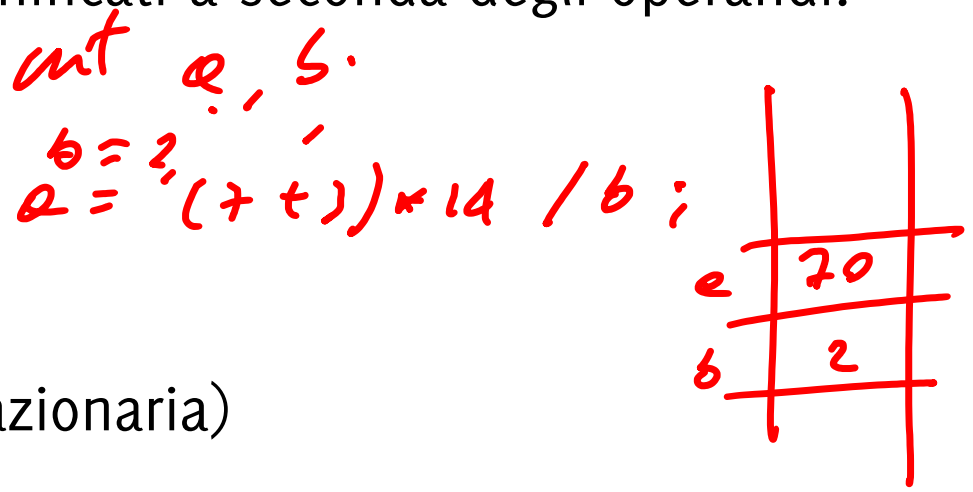
```
int a,b;
float c;
c = a / b;
```

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

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

oppure

```
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 tra Variabili Intere

- 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;`
- È un operatore che permette di verificare la divisibilità tra numeri

Operatori Aritmetici tra Variabili Intere

- 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;`
- Esempi:

```
int a = 11; int b = 4; int c;
a = a + 1;
c = a / b;
```

Operatori Aritmetici (cnt)

- 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;`
- Esempi:

```
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;
```

c = 2

Operatori Aritmetici (cnt)

- 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;`
- Esempi:

```
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`)

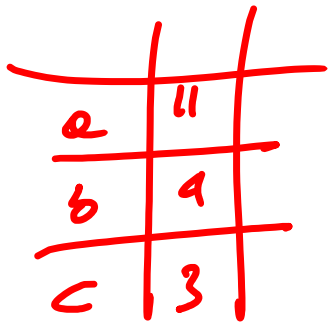
```
int a = 12; int b = 5; int c;
```

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

Operatori Aritmetici (cnt)

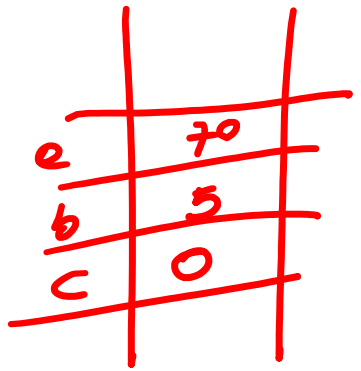
- Un nuovo operatore aritmetico: resto della divisione intera, o ***modulo*** `%`
  - es. `17%5` vale `2`, `15%5` vale `0`
- Segue che, con `a` e `b` interi: `a = (a/b)*b + a % b;`
- Esempi:

```
int a = 11; int b = 4; int c;
c = a % 2;
```




```
int a = 70; int b = 5; int c;
c = a % (b + 2);
```

7



Operatori Aritmetici (cnt)

- 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;`
- Esempi:

```
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` :

Operatori Aritmetici (cnt)

- 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;`
- Esempi:

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

int a=11, b=a, c;

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

```
int a = 70; int b = 5; int c;
```

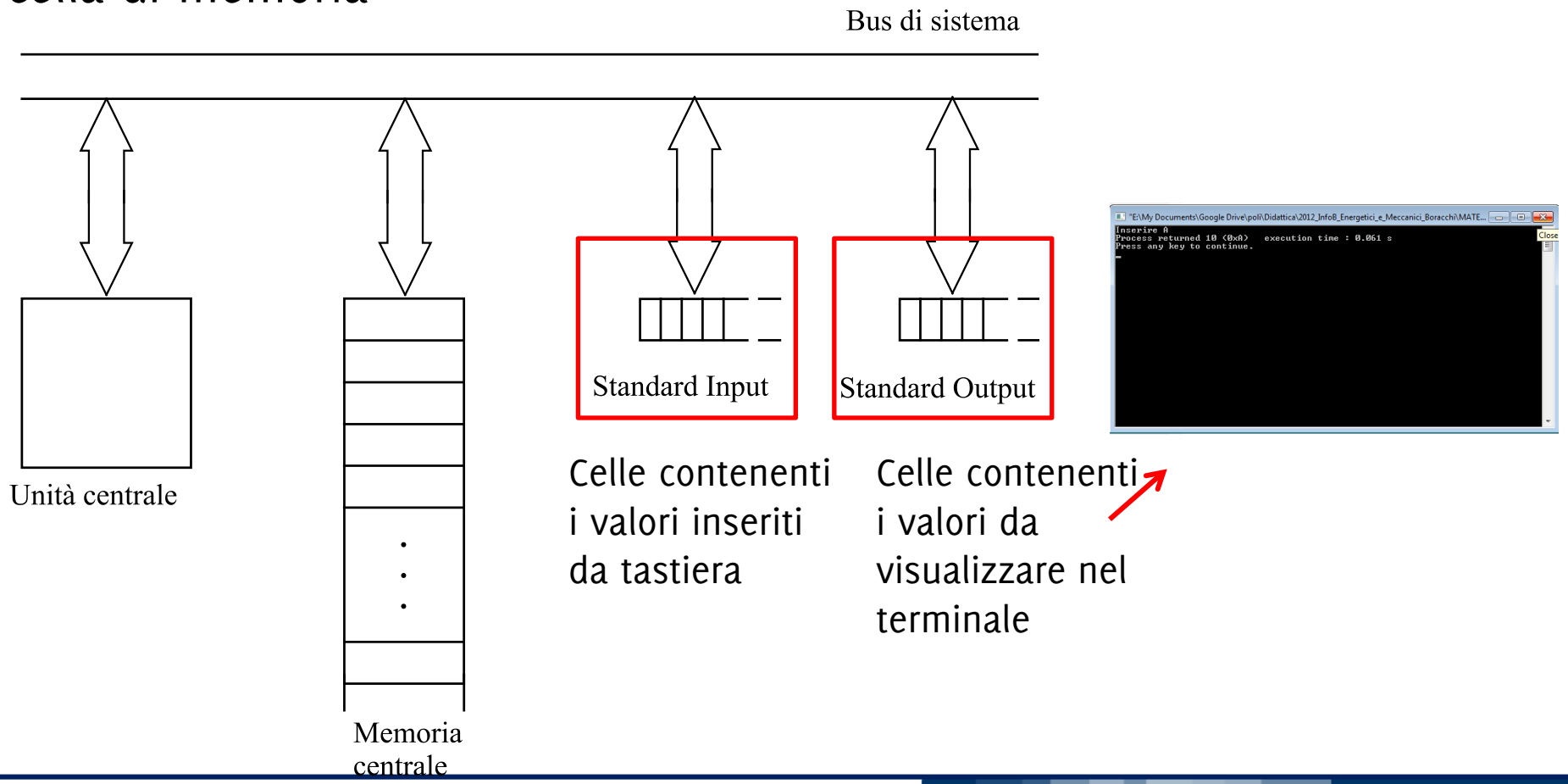
`c = a % (b + 2);` (viene scritto in `c` il valore `0`)

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

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

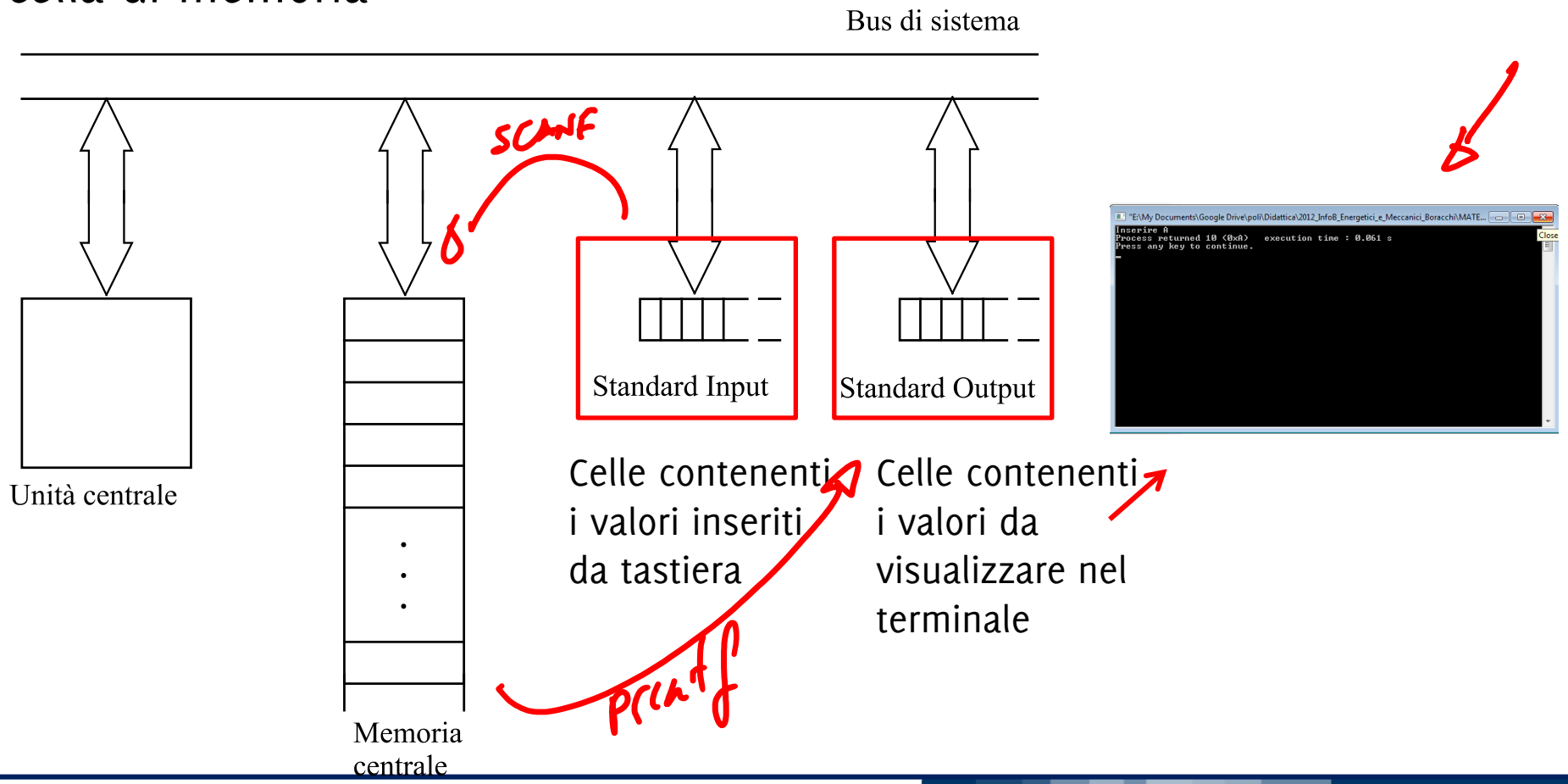
Istruzioni Ingresso ed Uscita

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



Istruzioni Ingresso ed Uscita

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



Istruzioni Ingresso ed Uscita (cnt)

- **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**

Istruzioni Ingresso ed Uscita (cnt)

- **Scrittura su schermo**: sintassi semplificata

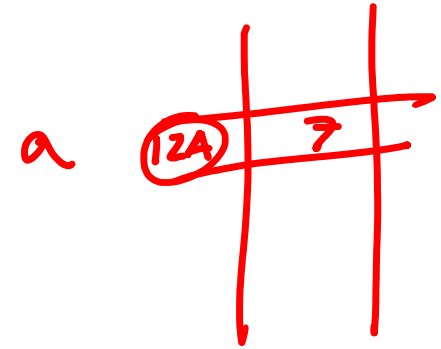
```
printf (stringaControllo);
```

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);
```

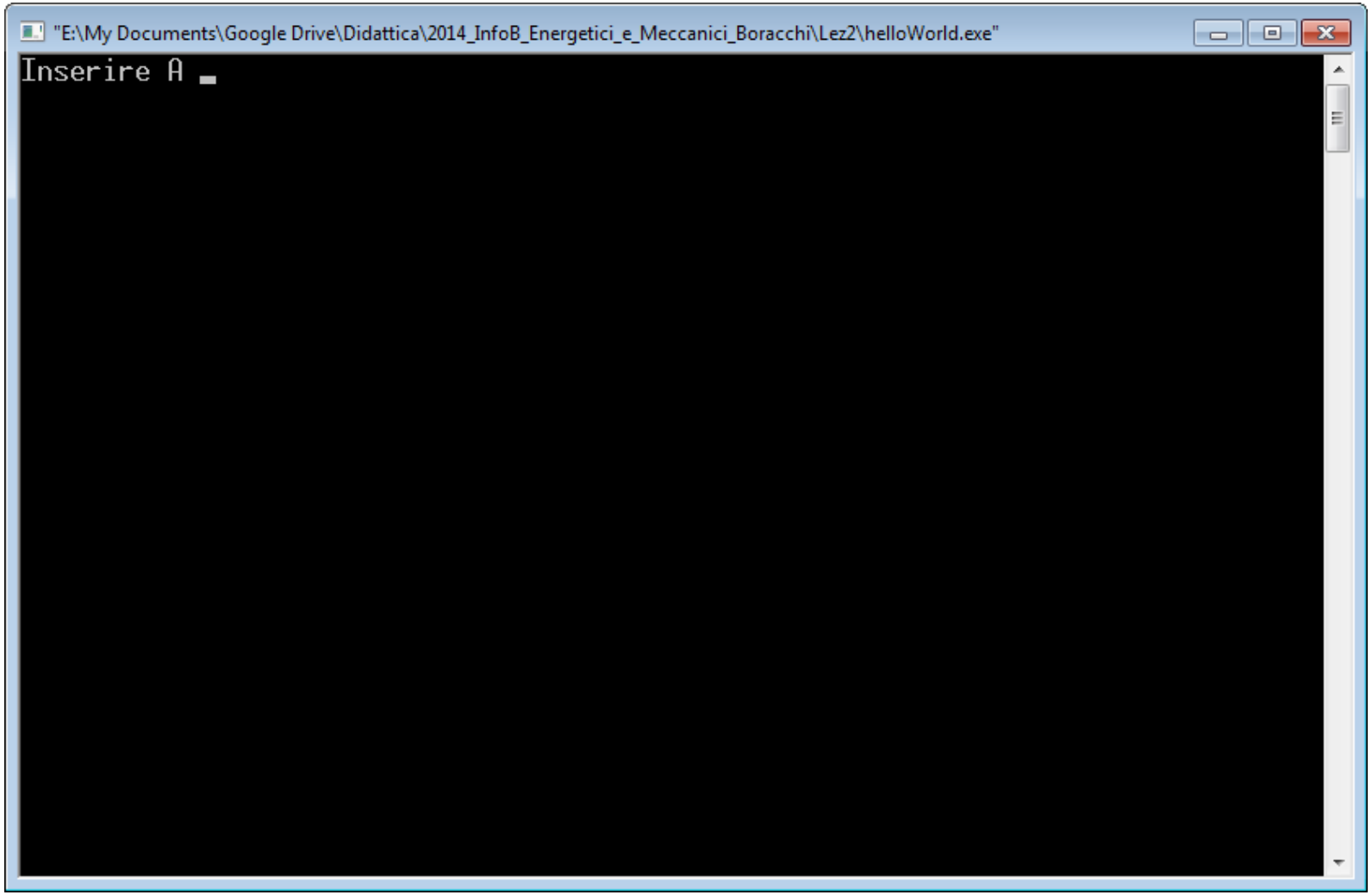
a è una variabile intera dichiarata in precedenza



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**

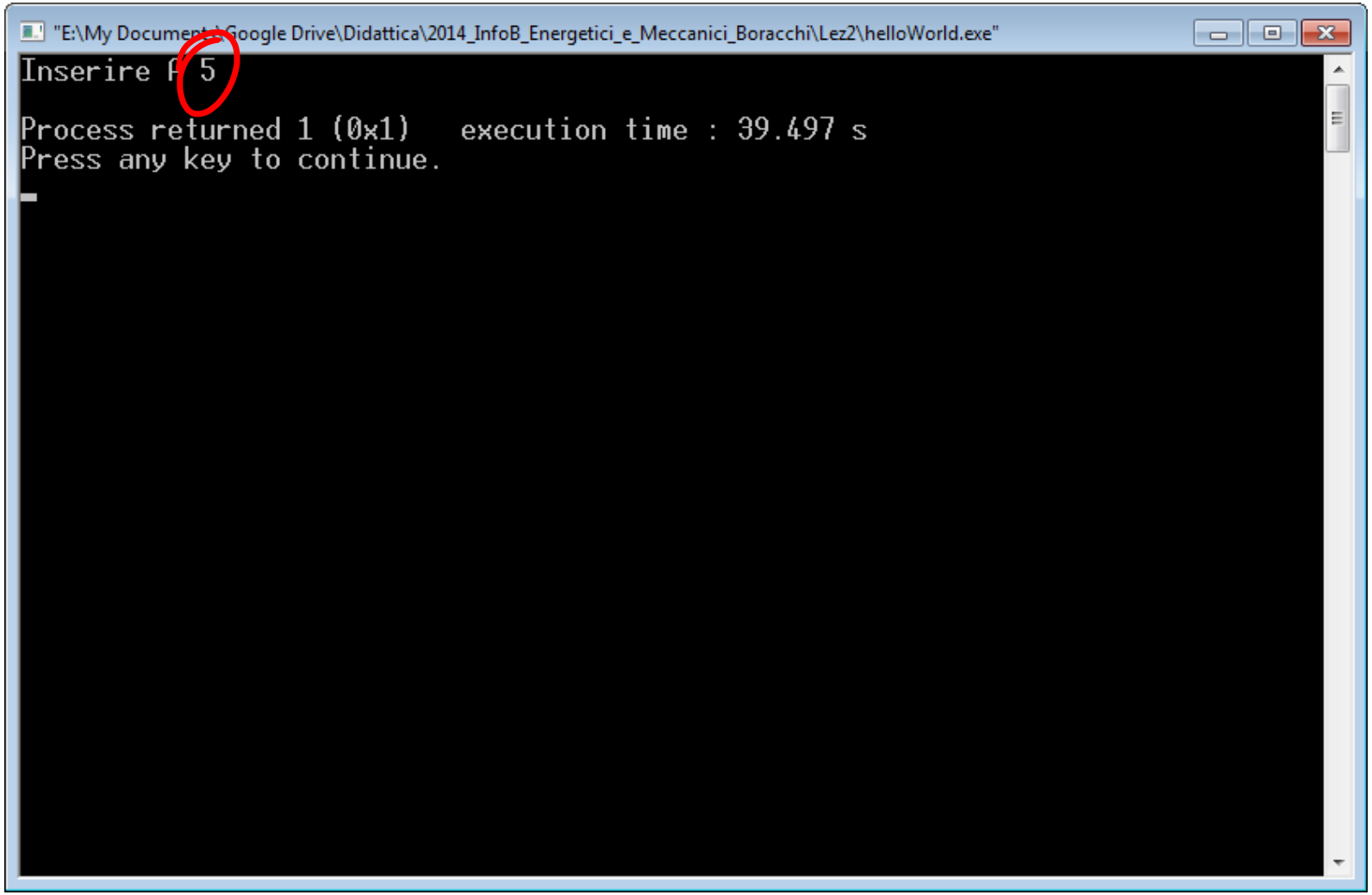
Esempio di Schermata di dialogo per I/O

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



Esempio di Schermata di dialogo per I/O

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



La Sequenzialità

La Sequenza di Istruzioni

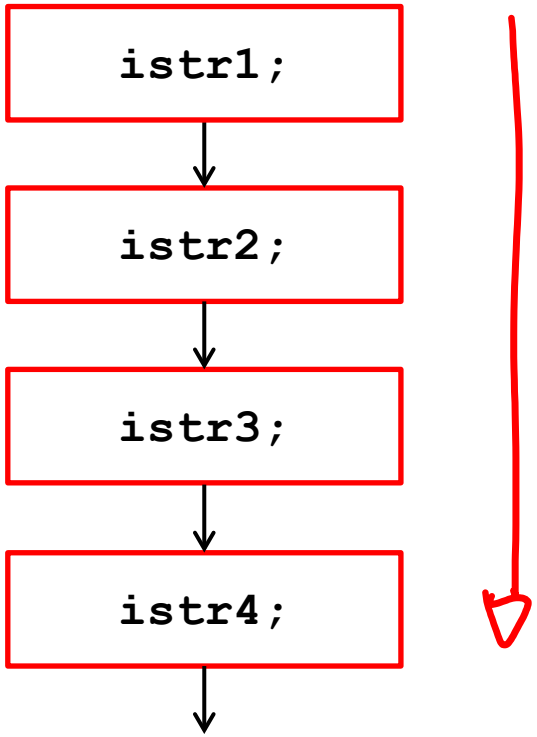
In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale,** dalla prima all'ultima

Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

```
1. Istruzione1;
2. Istruzione2;
3. ….
4. IstruzioneN;
```

I programmi C finora, sequenze di istruzioni

Diagramma di flusso



codice C

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

La Sequenza di Istruzioni

In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale,** dalla prima all'ultima

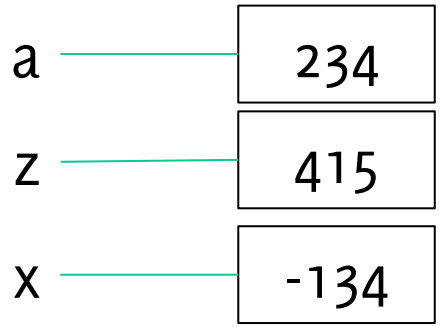
Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

```
1. Istruzione1;
2. Istruzione2;
3. ….
4. IstruzioneN;
```

- *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

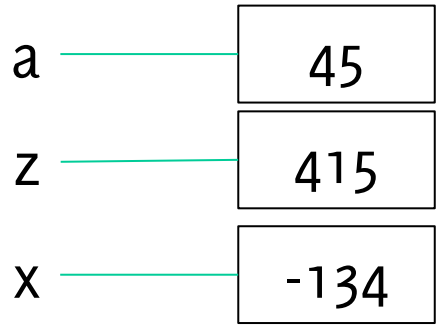
Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

```
1. Istruzione1;
2. Istruzione2;
3. ….
4. IstruzioneN;
```

- *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

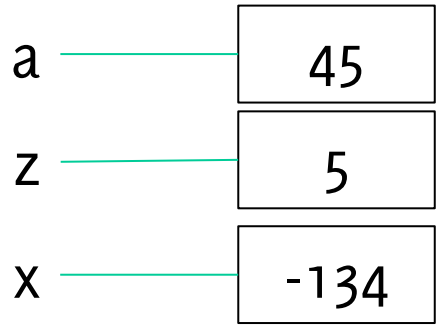
Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

```
1. Istruzione1;
2. Istruzione2;
3. ….
4. IstruzioneN;
```

- *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

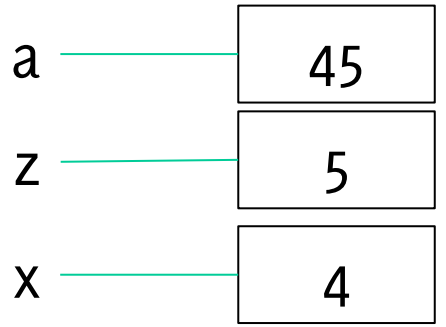
Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

1. `Istruzione1;`
2. `Istruzione2;`
3. ….
4. `IstruzioneN;`

- *Es:*

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

- Stato della memoria

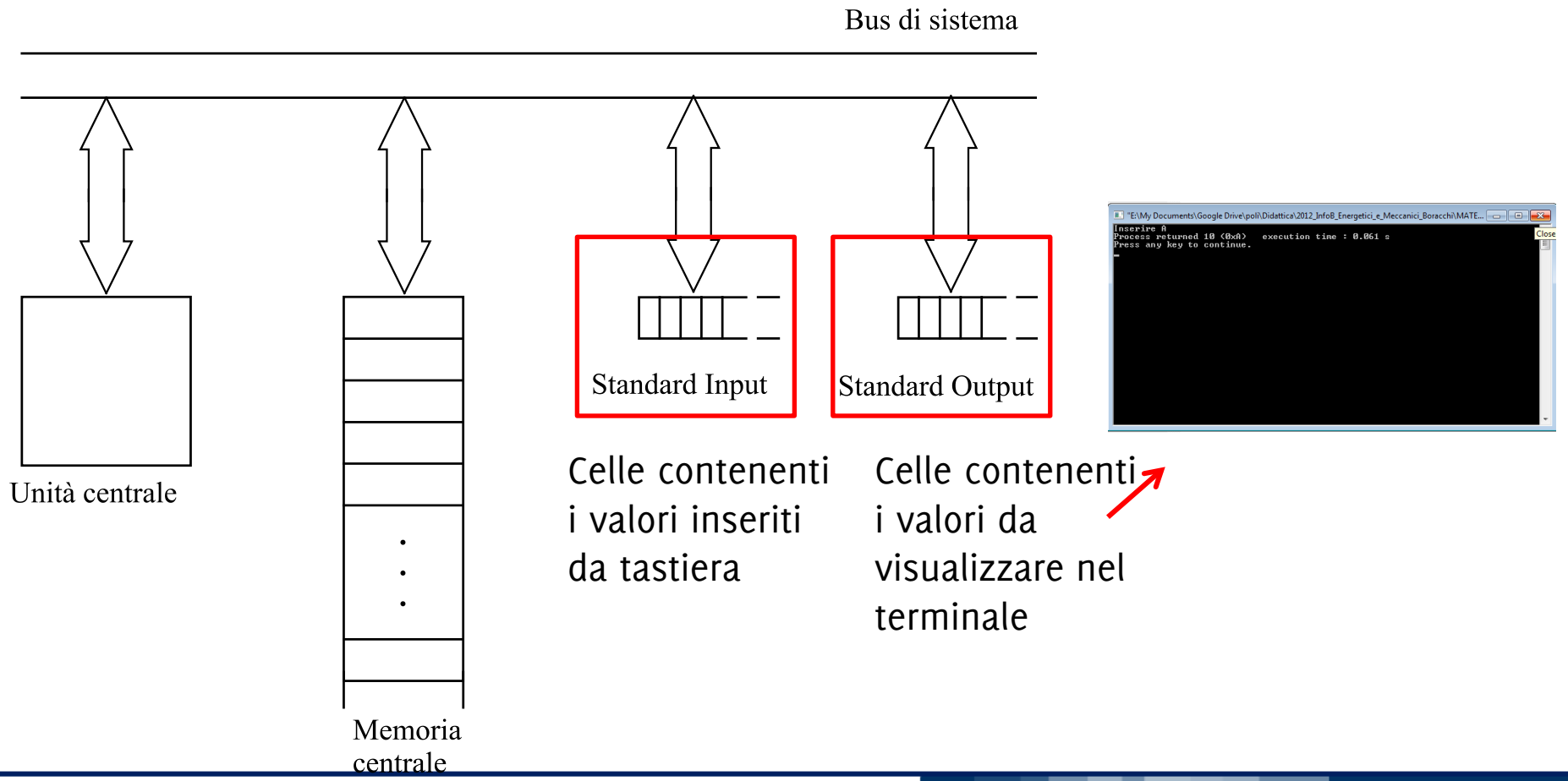


Un po' più di dettagli su I/O

Istruzioni Ingresso ed Uscita

`printf`: scrittura su Standard Output (uscita)

`scanf`: lettura da Standard Input e copia in una cella di memoria (ingresso)



Scrittura su Standard Output: `printf`

Esempio:

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

Sintassi:

```
printf (stringaControllo);
```

- ***stringaControllo*** sequenze di caratteri (i.e., stringa) delimitata da doppi apici " ". Possono essere
  - **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);
```

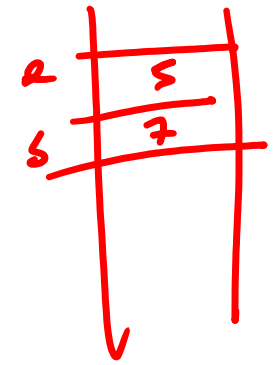
- ***stringaControllo*** può contenere
  - **caratteri di stampa** (**normali** o **speciali**)
  - **caratteri di conversione** (segnaposto, convertono valori di variabili in caratteri per la stampa)
- ***elementiStampa*** elenco di **variabili**, **espressioni composte,** o costanti separati da virgole

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

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

> 5 + 7 = 12

Ogni elemento di ***elementiStampa*** viene convertito in caratteri e associato ai caratteri di conversione in ***stringaControllo*** in **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)

Non occorre specificare gli apici singoli per caratteri all'interno di stringhe (delimitate da apici doppi)

Scrittura su Standard Output: `printf`

Esempi:

```
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`

Esempi:

```
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).

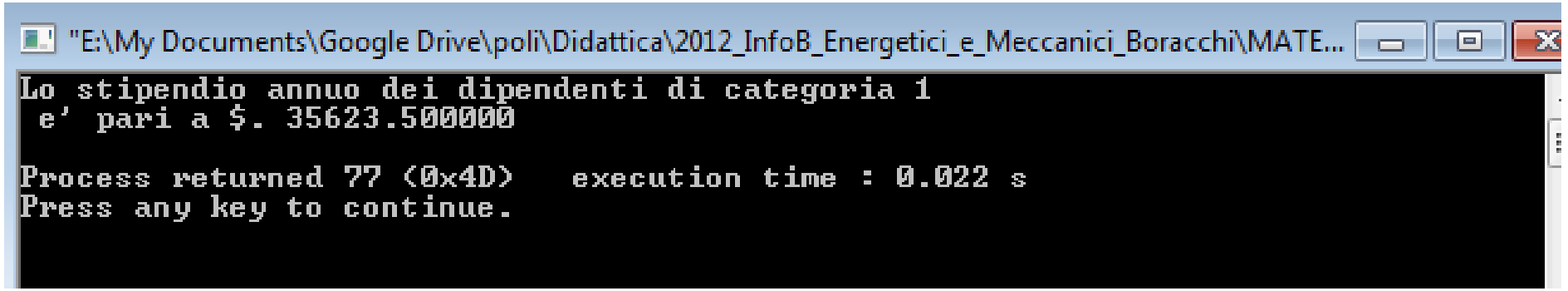
Scrittura su Standard Output: `printf`

Esempi:

```
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`

Esempi:

Così si indica il carattere ASCII

```
char iniz_nome = 'G';

char iniz_cognome = 'B';

printf("Questo programma è stato scritto da \n%c%c\n\nBuon
lavoro!\n", iniz_nome, iniz_cognome);
```

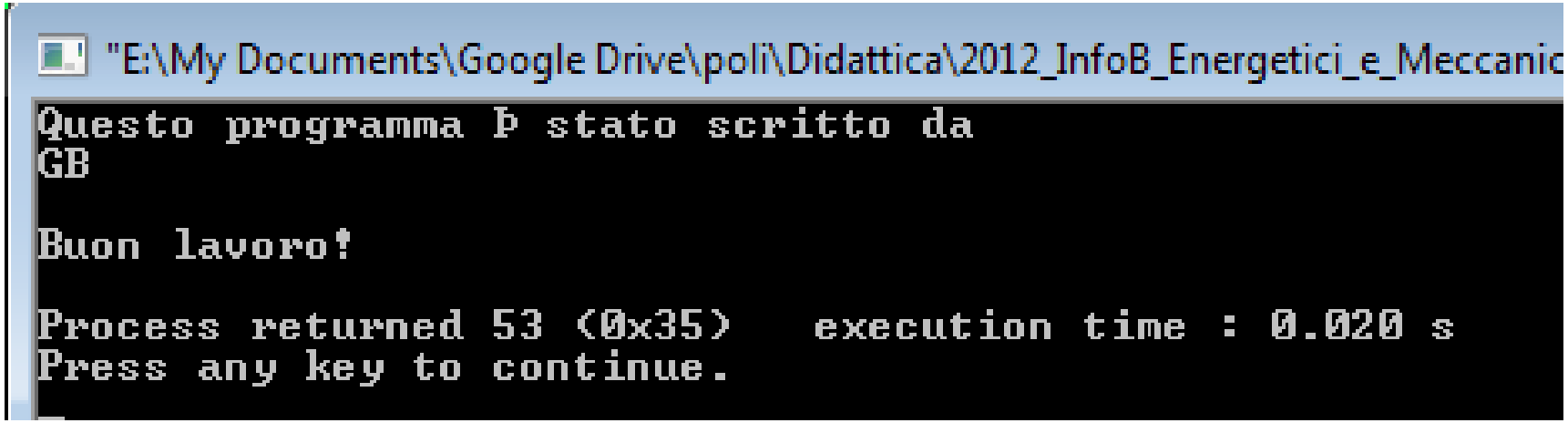
Scrittura su Standard Output: `printf`

Esempi:

```
char iniz_nome = 'G';

char iniz_cognome = 'B';

printf("Questo programma è stato scritto da \n%c%c\n\nBuon
lavoro!\n", iniz_nome, iniz_cognome);
```



Scrittura su Standard Output: `printf`

Esempi:

```
char iniz_nome = 'G';

char iniz_cognome = 'B';

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`
- **N.B:** non esiste un tipo di variabile built-in per contenere una stringa, occorrerà usare gli array.

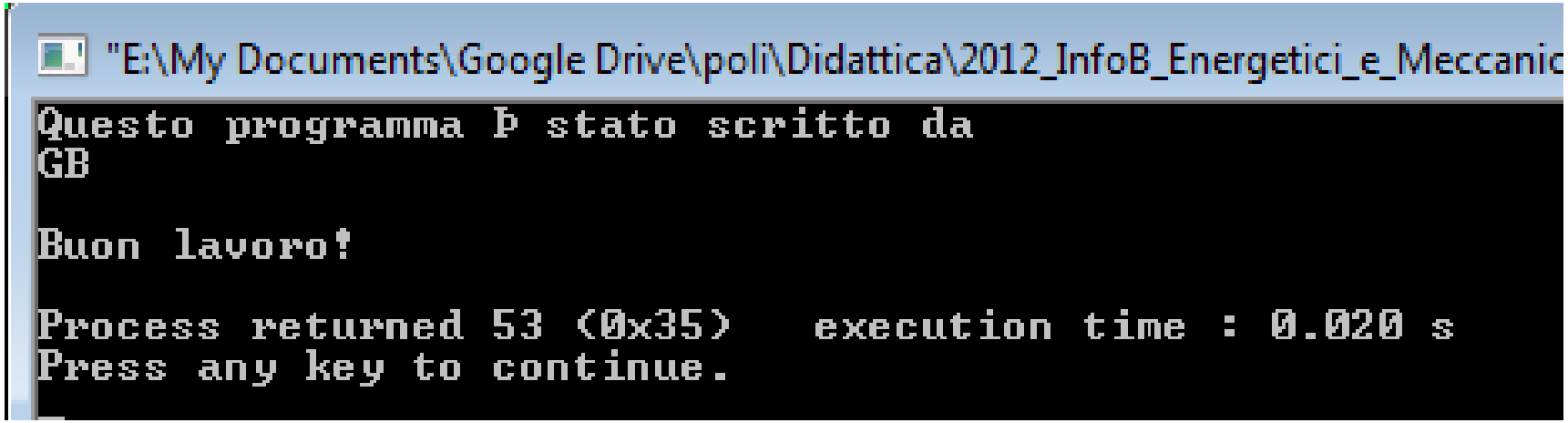
Scrittura su Standard Output: `printf`

Esempi:

```
char iniz_nome = 'G';

char iniz_cognome = 'B';

printf("%s\n%c%c\n\n%s\n", "Questo programma è stato
scritto da", iniz_nome, iniz_cognome, "Buon lavoro!");
```



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

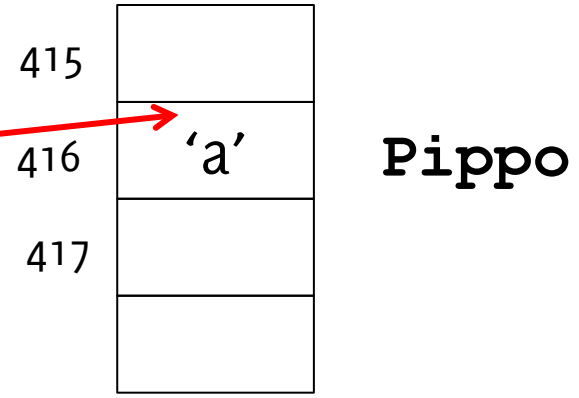
L'indirizzo di una variabile si ottiene anteponendo l'operatore **&** al nome della variabile

```
char Pippo;

Pippo = 'a';

Pippo

&Pippo
```



Nella pratica l'indirizzo delle variabili è scritto in formato esadecimale.

Lettura da Standard Input: `scanf`

Esempi di acquisizione di diversi tipi

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

Acquisizioni multiple sono **possibili ma sconsigliate!**

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

E qualche programma più elaborato

Struttura di un programma C

```
/* eseguire la somma di due
numeri inseriti dall'utente*/
# include<stdio.h>
int 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);
    return 0;
}
```

Parte **dichiarativa** del programma

- Dichiara le variabili e le costanti utilizzate dal programma

- Facilita la diagnostica dei programmi, typos e autocompletamento

Struttura di un programma C

```c
/* eseguire la somma di due
numeri inseriti dall'utente*/
# include<stdio.h>
int 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);
    return 0;
}
```

parte dichiarativa

Parte **dichiarativa** del programma

- Dichiara le variabili e le costanti utilizzate dal programma

- Facilita la diagnostica dei programmi, typos e autocompletamento

Struttura di un programma C

```
/* eseguire la somma di due
numeri inseriti dall'utente*/
# include<stdio.h>
int 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);
    return 0;
}
```

Parte **esecutiva** del programma, contiene le istruzioni del programma. In questo caso:

- I/O
- Assegnamento

NB: l'incolonnamento dei programmi (i.e. tab e spazi) sono irrilevanti per il compilatore, ma facilitano la comprensione dei codici

Esempio di programmi

Scrivere un programma che prende in ingresso una temperatura in Fahrenheit e la trasforma in Celsius

$$C = 5/9 * (F - 32)$$

Esempio di programmi

Scrivere un programma che prende in ingresso una temperatura in Fahrenheit e la trasforma in Celsius

$$C = 5/9 * (F - 32)$$

```
// 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);
}
```

Esempio di programmi

Scrivere un programma che prende in ingresso un prezzo in euro e restituisce il numero minimo di banconote utilizzando solo pezzi da 50, 20 e 5 euro. Indicare anche la moneta rimanente.

Esempio di programmi

Scrivere un programma che prende in ingresso un prezzo in euro e restituisce il numero minimo di banconote utilizzando solo pezzi da 50, 20 e 5 euro. Indicare anche la moneta rimanente.

```
#include<stdio.h>
void main()
{
 int prezzo, rimanente, n50, n20,n5;
 printf("Inserisci il prezzo:\n");
 scanf("%d",&prezzo);
 n50 = prezzo / 50;
 rimanente = prezzo % 50;
 n20 = rimanente / 20;
 rimanente = rimanente % 20;
 n5 = rimanente / 5;
 rimanente = rimanente % 5;
 printf("Banconote 50:%d\n",n50);
 printf("Banconote 20:%d\n",n20);
 printf("Banconote 5: %d\n",n5);
 printf("Monete: %d\n",rimanente);

}
```

TODO

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

Esempio

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.

Esempio

```
# 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.

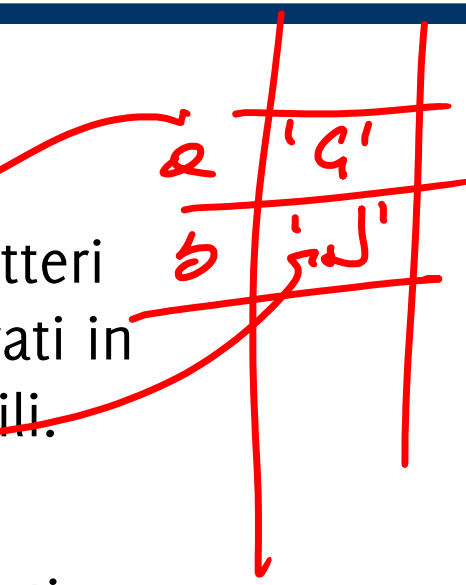
Esempio

```
# 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.



Esempio

```
# 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 buffer di ingresso, lo standard input.

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

Vi ricordate?
Algoritmo per invertire il contenuto di A e B

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

Algoritmo per scambiare i valori di due variabili A e B (con le variabili a volte non occorre il bicchiere C)

Linguaggio C: Costrutto Condizionale

Istruzioni composta: `if`

Costrutto Condizionale: `if`, la sintassi

- Il costrutto **condizionale** permette di eseguire alcune istruzioni a seconda del valore di un'espressione booleana a runtime
- `if,else` keywords
- **`expression`** espressione booleana (vale 0 o 1)
- **`statement`** è la sequenza di istruzioni da eseguire (corpo) quando **`expression`** è vera (nel caso di **`statement0`** quando è falsa)
- se **`statement`** contiene più istruzioni, va delimitato tra { }
- **NB** indentatura irrilevante

```
if(expression)
        statement
```

```
if(expression)
        statement1
else
        statement0
```

Costrutto Condizionale: `if`, l'esecuzione

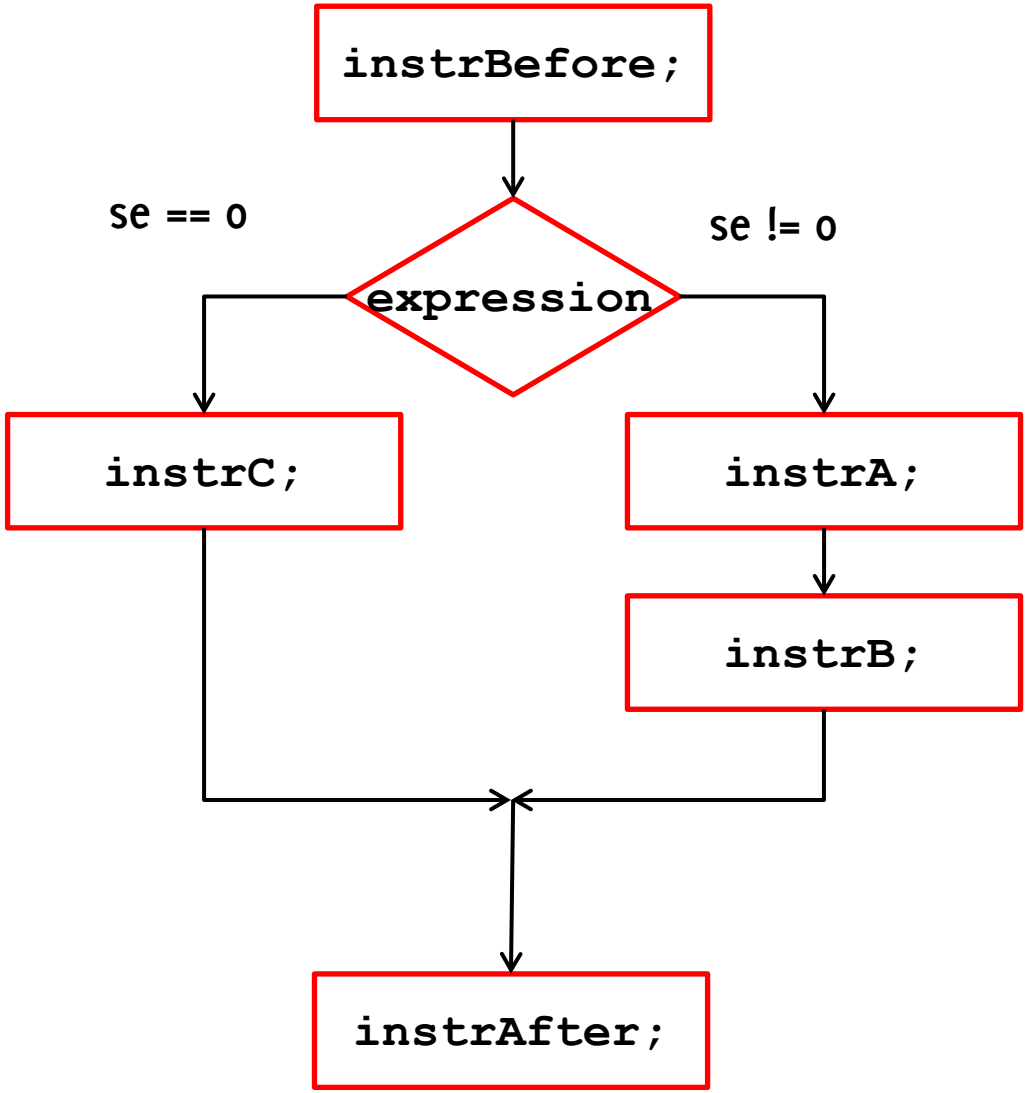
1. Terminata **`instrBefore`**, valuto **`expression`**,
2. Se **`expression`** è vera (**`!=0`**), allora eseguo **`statement1`**, altrimenti eseguo **`statement0`**. (se è presente **`else`**)
3. Terminato lo statement dell'**`if`** , procedi con **`instrAfter`**, la prima istruzione fuori dall'**`if`**

N.B. **`else`** è opzionale

N.B **`if(expression)`** **non** richiede il **`;`** perché l'istruzione non termina dopo **`)`**

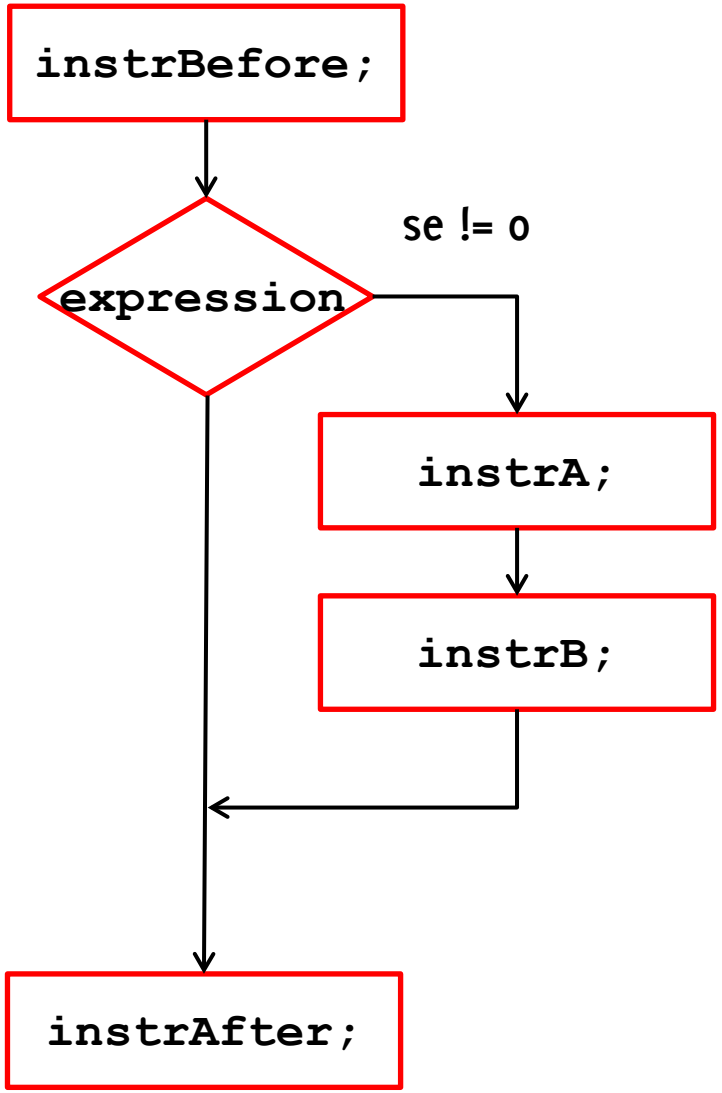
```
instrBefore;
if(expression)
        statement1;
else
        statement0;
instrAfter;
```

Costrutto Condizionale: `if`, l'esecuzione




```
instrBefore;
if(expression)
        {
        instrA;
        instrB;
        }
else
        instrC;
instrAfter;
```

Costrutto Condizionale: `if`, l'esecuzione




```
instrBefore;
if(expression)
        {
        instrA;
        instrB;
        }
instrAfter;
```

Esempio

```
//N.B: incolonnamento codice irrilevante!
if (x % 7 == 0)
     printf("%d multiplo di 7" , x);
else
     printf("%d non multiplo di 7" , x);

//si può fare senza else?
```

Esempio

```
//N.B: incolonnamento codice irrilevante!
if (x % 7 == 0)
     printf("%d multiplo di 7" , x);
else
     printf("%d non multiplo di 7" , x);

//senza else.
printf("%d " , x);
if (x % 7 != 0)
     printf("non "); // { printf("non "); }
printf(" multiplo di 7");
```

`if` Annidati

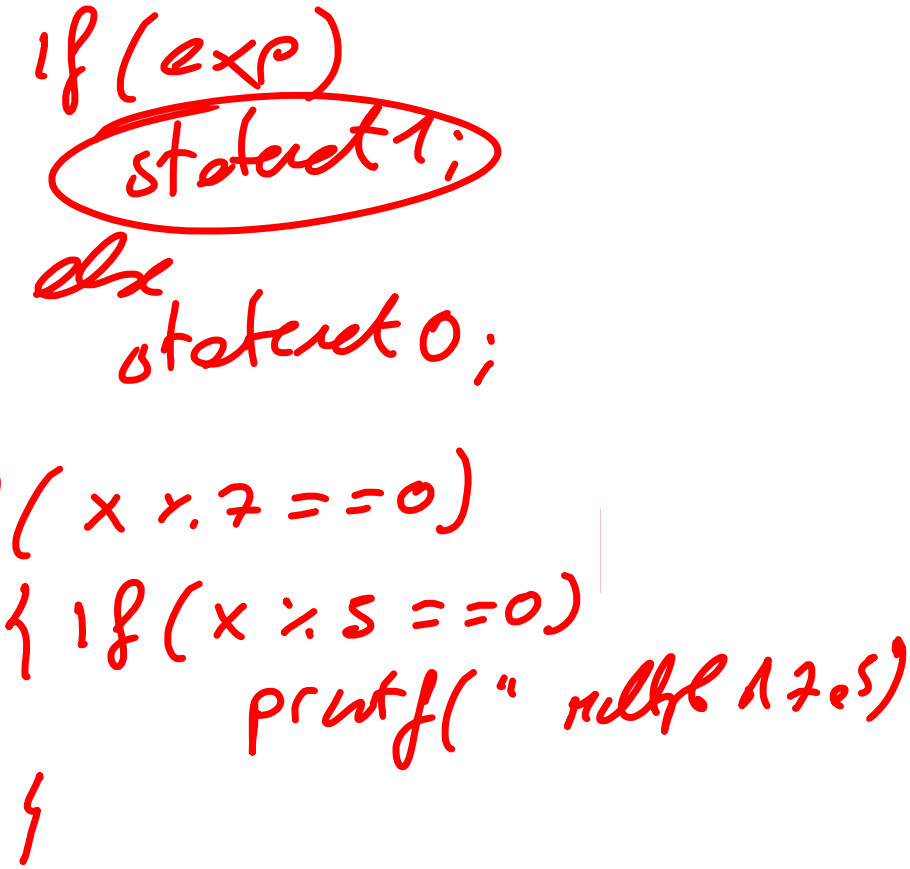
Il corpo di un **`if`** (cioè gli **`statement`**) può a loro volta contenere costrutti altri **`if:`** si realizzano quindi istruzioni condizionali **annidate**

```
instrBefore;
if(expr1)
 if(expr2)
      instrA;
  else
      instrD;
else
 instrC;
instrAfter;
```

`if` Annidati

Il corpo di un `if` (cioè gli **`statement`**) può a loro volta contenere costrutti altri `if`: si realizzano quindi istruzioni condizionali **annidate**

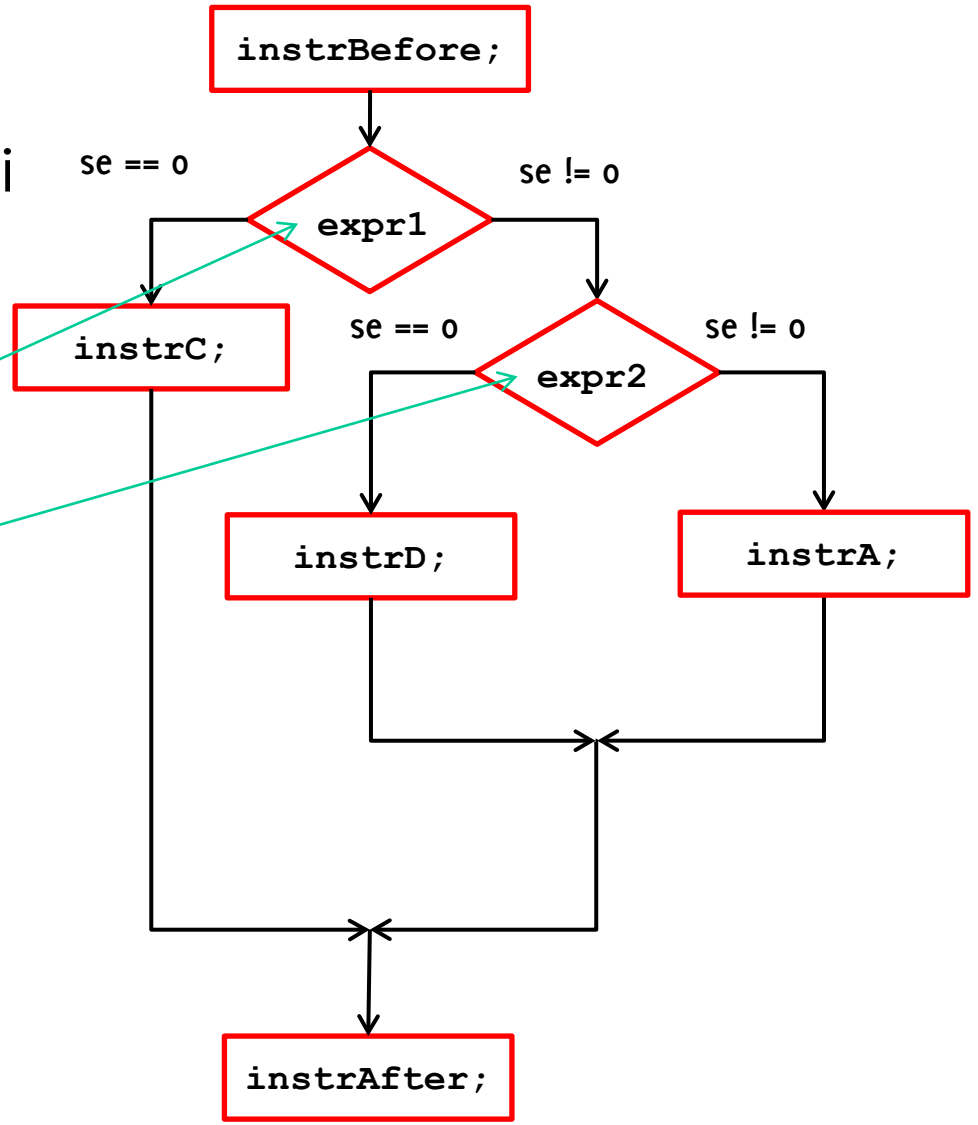
```
instrBefore;
if(expr1)
 if(expr2)
      instrA;
  else
      instrD;
else
 instrC;
instrAfter;
```



`if` Annidati

Il corpo di un **`if`** (cioè gli **`statement`**) può a loro volta contenere costrutti altri **`if:`** si realizzano quindi istruzioni condizionali **annidate**

```
instrBefore;
if(expr1)
 if(expr2)
      instrA;
  else
      instrD;
else
 instrC;
instrAfter;
```



`if` Annidati

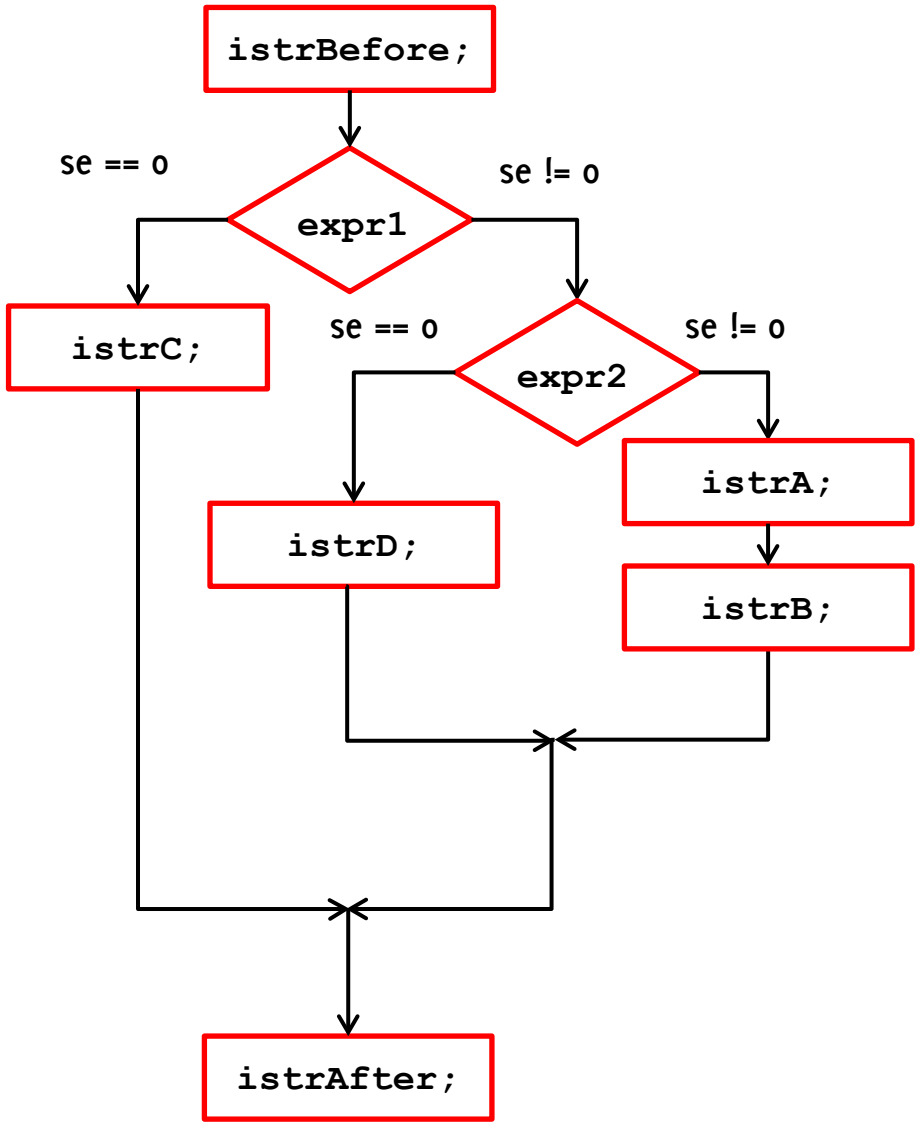
Quando il corpo di un if contiene più di un'istruzione è necessario usare parentesi.

```
instrBefore;
if(expr1)
 if(expr2)
      {instrA;
       instrB;}
  else
      instrD;
else
 instrC;
instrAfter;
```

if Annidati

Quando il corpo di un if contiene più di un'istruzione è necessario usare parentesi.

```
instrBefore;
if(expr1)
 if(expr2)
      {instrA;
       instrB;}
  else
      instrD;
else
 instrC;
instrAfter;
```



`if` Annidati

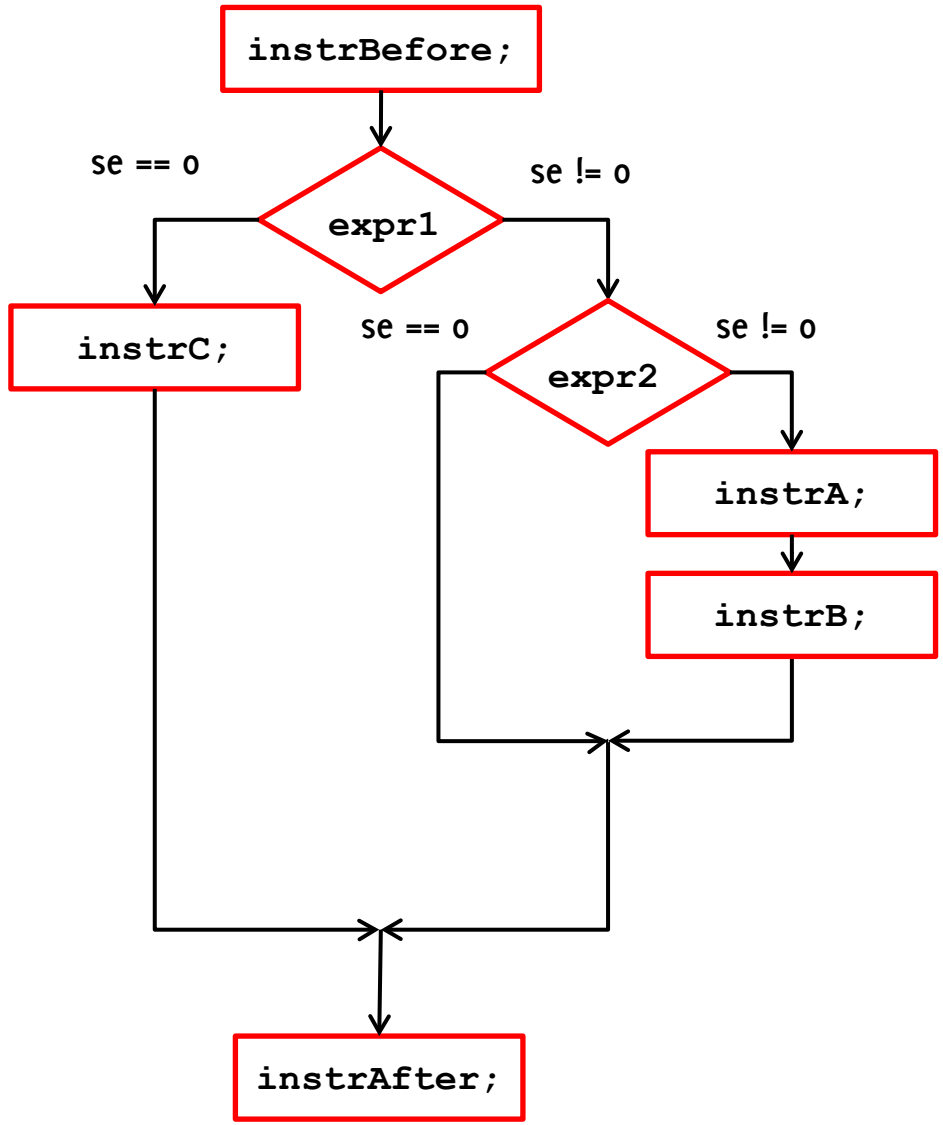
L'uso delle parentesi serve per determinare quale **`else`** associare a quale **`if`**.

```
instrBefore;
if(expr1)
{ if(expr2)
      {instrA;
       instrB;}
}
else
 instrC;
instrAfter;
```

if Annidati

L'uso delle parentesi serve per determinare quale **else** associare a quale **if**.

```
instrBefore;
if(expr1)
{ if(expr2)
      {instrA;
       instrB;}
}
else
 instrC;
instrAfter;
```



`if` Annidati

Le istruzioni condizionali possono essere annidate, inserendo un ulteriore **`if`** all'interno di **`statement1`** o **`statement0`**

Esempio

```
if ( x % 5 == 0 )
   if (x % 7 == 0 )
       printf("x multiplo di 5 e anche di 7");
   else
       printf("x multiplo di 5 ma non di 7");
else
    printf("x non multiplo di 5");
```

Regole per `if` annidati

Regola: In caso di costrutti annidati, ed in assenza di parentesi che indichino diversamente, ogni **`else`** viene associato all' **`if`** più vicino.

```
if ( x % 5 == 0 )
      if (x % 7 == 0 )
            printf("x multiplo di 5 e anche di 7");
      else
            printf("x multiplo di 5 ma non di 7");
```

Regole per `if` annidati

Regola: In caso di costrutti annidati, ed in assenza di parentesi che indichino diversamente, ogni **`else`** viene associato all' **`if`** più vicino.

```
if ( x % 5 == 0 )
      {
      if (x % 7 == 0 )
          printf("x multiplo di 5 e anche di 7");
      }
else
      printf("x non multiplo di 5");
```

Esempio

Scrivere un programma che, inserito un intero positivo, determina se corrisponde ad un anno bisestile

- Un anno è bisestile se
  - è multiplo di 4 ma non di 100
  - oppure se è multiplo di 400

La Catena di Programmazione

Come si passa dal codice C al programma eseguibile?

La catena di programmazione dei linguaggi compilati

Si parte dalla codifica di un algoritmo in un **codice sorgente**

- fatta tramite un linguaggio simbolico
  - di basso livello (Assembler)
  - o di alto livello (C, Fortran, …)

Si arriva a generare un programma scritto in codice macchina, chiamato programma eseguibile

1. Videoscrittura (editing)

- Il testo del programma sorgente, costituito da una sequenza di caratteri, viene composto e modificato usando uno specifico programma: l'editor
- Così otteniamo un File Programma Sorgente memorizzato in memoria di massa in un file di testo di nome:
  - XXX.asm per programmi in assembler
  - XXX.c per programmi in C
  - XXX.cpp per programmi in C++
  - … …

2. Traduzione

- Linguaggio di alto livello $\Rightarrow$ Linguaggio macchina (compilatore)
- Durante questa fase si riconoscono i simboli, le parole e i costrutti del linguaggio:
  - eventuali messaggi diagnostici segnalano errori lessicali e sintattici
- Si genera la forma binaria del codice macchina corrispondente: a partire dal File Programma Sorgente si genera un File Programma Oggetto, cioè in un file binario di nome XXX.obj

3. Collegamento (linking)

Il programma collegatore (linker) deve collegare fra loro il file oggetto e i sottoprogrammi richiesti (es. le funzioni di C)

I sottoprogrammi sono estratti dalle librerie oppure sono individuati tra quelli definiti dal programmatore (nel qual caso si trovano anch'essi nel file oggetto)

Si rendono globalmente coerenti i riferimenti agli indirizzi dei vari elementi collegati

Si genera un File Programma Eseguibile, un file binario che contiene il codice macchina del programma eseguibile completo, di nome XXX.exe

Messaggi di errore possono essere dovuti ad errori nel citare i nomi delle funzioni da collegare

Il programma sarà effettivamente eseguibile solo dopo che il contenuto del file sarà stato caricato nella memoria di lavoro (centrale) del calcolatore (a cura del Sistema Operativo)

4. Caricamento (loading)

Il caricatore (loader) individua una porzione libera della memoria di lavoro e vi copia il contenuto del file XXX.exe

- Eventuali messaggi rivolti all'utente possono segnalare che non c'è abbastanza spazio in memoria

5. Esecuzione

- Per eseguire il programma occorre fornire in ingresso i dati richiesti e in uscita riceveremo i risultati (su video o file o stampante)
- Durante l'esecuzione possono verificarsi degli errori (detti "errori di run-time"), quali:
  - calcoli con risultati scorretti (per esempio un overflow)
  - calcoli impossibili (divisioni per zero, logaritmo di un numero negativo, radice quadrata di un numero negativo,….)
  - errori nella concezione dell'algoritmo (l'algoritmo non risolve il problema dato)
  - Tutti gli esempi citati si riferiscono ai cosiddetti errori semantici

Nel caso del C le fasi sono sei

- Videoscrittura
  - produzione del programma, svolta dal programmatore tramite un programma di videoscrittura (editor)
- Pre-compilazione (pre-processing)
  - svolta da un programma detto preprocessore
- Traduzione (compilazione)
  - svolta dal compilatore (compiler)
- Collegamento (linking)
  - svolto dal collegatore (linker)
- Caricamento (loading)
  - svolto dal caricatore (loader)
- Esecuzione
  - a cura del Sistema Operativo