Tecniche della Programmazione, lez. 3

- Introduzione allo sviluppo ed esecuzione di programmi

Sintassi …

Un programma è una sequenza di istruzioni, scritte nel linguaggio di programmazione in uso (Per noi il *C*).

E` un lungo paragrafo, costituito da tante frasi, una dopo l'altra (le istruzioni).

Per scrivere bene una frase bisogna seguire delle regole, questo vale per qualsiasi linguaggio …

La **SINTASSI** è la grammatica da seguire per scrivere le frasi (le istruzioni)

```
deltaQuadro = b*b – 4*a*c;
```

SI

Segue le regole grammaticali

```
Delta Quadro = bb – ; 4a*c
```

NO

Non segue le regole grammaticali

… e Semantica

Una frase scritta grammaticalmente bene può avere un significato, oppure no:

Lucilla mangia la mela VS. La mela mangia Lucilla

La **SEMANTICA** è il significato dell'istruzione … e anche il significato deve essere giusto, per avere un programma corretto

```
primoNum = 16;
```

SI (se primoNum e` un simbolo associabile ad un intero)

```
primoNum= "miao";
```

NO (nella stessa ipotesi)

```
stampaIntero(47);
```

SI (probabilmente)

```
stampaIntero(4a7.6);
```

NO

```
sqrt(47.1);
```

:)

```
sqrt(marco)
```

boh, dipende da cosa e` marco

Memento

Quando un programma viene eseguito,
viene eseguita la sequenza delle sue istruzioni,
una istruzione alla volta … secondo la sequenza

finché il programma finisce
oppure va in crash

Di solito noi scriviamo da sinistra a destra e dall'alto verso il basso …
e l'esecutore delle istruzioni si adegua: quello è l'ordine di esecuzione delle istruzioni

MEMENTO (gia` visto ...)

-ANALISI: QUALI dati? FATTI come (struttura)?

- **INPUT** numeri reali (double)
- **OUTPUT** numero reale
- **dati per calcoli intermedi** ...
- idea: **prod ← b*h**
  area ← prod/2

SINTESI ... =

1) prendere da INPUT (LETTURA) i valori da associare a b e h 1

2) calcolare prod (ASSEGNAZIONE) 2

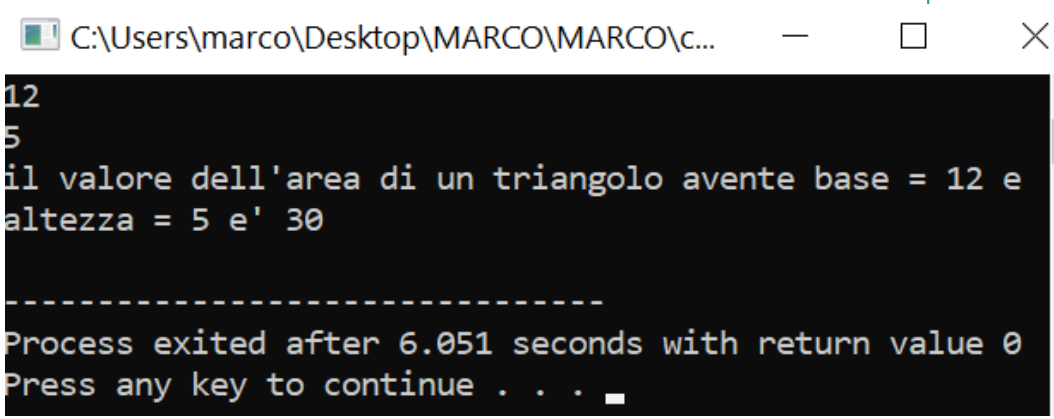
3) calcolare area (ASSEGNAZIONE) 3

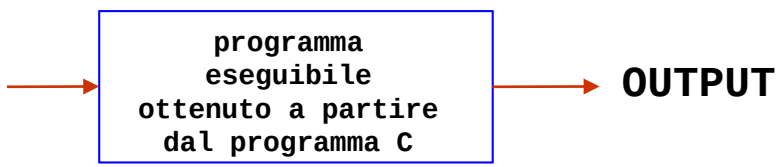
4) fornire in OUTPUT (SCRITTURA) il valore di area 4

Sviluppo di un programma

PROGRAMMAZIONE ... =

```
#include<stdio.h>
int main() {
  double b, h;
  double area;
  double prod;
 scanf("%lf %lf", &b, &h);          1
 prod = b*h;                        2
 area = prod/2;                     3
 printf("il valore dell'area di un
        triangolo avente base = %g e
        altezza = %g e` %g\n", b, h,
        area);                      4
n 0;
```





ISTRUZIONE DI STAMPA

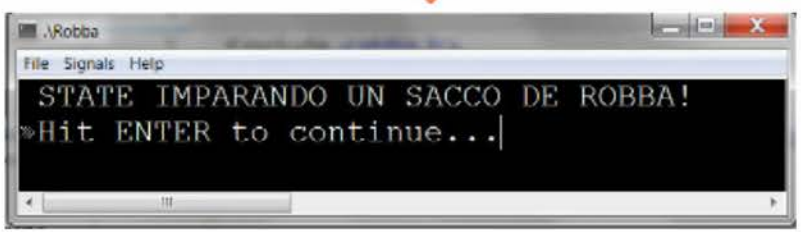
È POSSIBILE SCRIVERE UN'ISTRUZIONE DI STAMPA!

⇒ LA SEMANTICA DI UN'ISTRUZIONE DI STAMPA È CHE VIENE STAMPATO "QUALCOSA" (UNA FRASE, UN VALORE...) SULLO SCHERMO DELL'UTENTE

⇒ LA SINTASSI È `printf( );` ← QUA DENTRO CI VA QUELLO CHE VUOI STAMPARE FRA DOPPI APICI.

```
printf("STATE IMPARANDO UN SACCO DE ROBBA!");
```

QUELLO CHE SCRIVE IL PROGRAMMATORE



QUELLO CHE VEDE L'UTENTE CHE ESEGUE IL PROGRAMMA

Lettura, cioe` ricezione di dati da INPUT

i dati si "leggono" da INPUT; in Memoria (RAM) i dati memorizzati nelle locazioni si ottengono "accedendo" alle locazioni

LA FUNZIONE scanf PERMETTE DI LEGGERE VALORI IMMESSI DALL'UTENTE.

SINTASSI: scanf ("%FORMATO", &VARIABILE);

- %FORMATO INDICA IL FORMATO DEI VALORI DA LEGGERE (AD ES. %d PER LEGGERE UN INTERO) COME PER printf
- &VARIABILE INDICA L'INDIRIZZO DELLA MEMORIA AL QUALE SI TROVA LA VARIABILE

SEMANTICA: IL PROGRAMMA SI PONE IN ATTESA CHE L'UTENTE INTRODUCA UN VALORE CON FORMATO d. QUANDO L'UTENTE INTRODUCE UN VALORE, TALE VALORE VIENE MEMORIZZATO NELLA VARIABILE IL CUI INDIRIZZO È &VARIABILE.

sempre sulla lettura ...

ANALOGAMENTE AD UNA ISTRUZIONE DI ASSEGNAZIONE, UNA LETTURA HA COME EFFETTO QUELLO DI MEMORIZZARE UN VALORE IN UNA VARIABILE.

PER COMUNICARE ALL'UTENTE CHE DEVE IMMETTERE DEI VALORI, UNA ISTRUZIONE DI LETTURA È IN GENERE PRECEDUTA DA UNA ISTRUZIONE DI STAMPA.

```c
/* questo programma, ricevendo in INPUT i lati significativi di un rettangolo,
   ne calcola e stampa in OUTPUT l'area */
#include <stdio.h>

int main () {
  int primoLato, secondoLato,        /* i lati da ricevere via INPUT */
      area;                          /* risultato ... */

/* lettura dei dati */
  printf (" Oh utente, forniscimi gentilmente ..., che cosi' ci lavoro:\n ");
  scanf("%d %d", &primoLato, &secondoLato);

/* calcolo dell'area */
  area = primoLato*secondoLato;          /* calcolo */

  printf (" ... di lati %d e %d ha area %d\n", primoLato, secondoLato, area);

printf ("\nFINE");
return 0;
}
```

Istruzioni, funzioni e librerie

L'"istruzione di stampa" discussa prima consiste in realta` in una chiamata alla funzione printf.

"Chiamare una funzione" significa "richiedere l'esecuzione della funzione" ...

`printf()` e` una funzione non direttamente esistente nel linguaggio C, ma disponibile in un **modulo di programma** (**libreria**).

Per cui nel programma, all'inizio, dobbiamo indicare la libreria della quale ci serviremo (le cui funzioni chiameremo nel programma), attraverso una **direttiva include**

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

Istruzioni, funzioni e librerie

L'"istruzione di stampa" discussa prima consiste in realta` in una chiamata alla funzione printf.

"Chiamare una funzione" significa "richiedere l'esecuzione della funzione" ...

`printf()` e` una funzione non direttamente esistente nel linguaggio C. Per usarla bisogna indicare nel programma dove la sua definizione e il relativo codice eseguibile possono essere trovati.

Queste definizioni sono disponibili tramite la "**libreria** di input/output" `stdio.h`

> Esistono molte librerie, contenenti funzioni definite per gli scopi piu' diversi, ed utili nella costruzione dei programmi.
> Una libreria ("library") e` il **modulo software** (parte di un programma, predefinita e disponibile nell'ambiente di programmazione, che usiamo per fare i programmi).
> E` una collezione di programmi gia` fatti che possiamo usare nei nostri programmi ... possiamo anche scrivere una libreria per conto nostro, e poi usarla in un programma

Per cui nel programma, all'inizio, dobbiamo indicare la libreria della quale ci serviremo (le cui funzioni chiameremo nel programma), attraverso una **direttiva include**

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

SPAZI E RITORNI A CAPO

LO SPAZIO È UN CARATTERE COME TUTTI GLI ALTRI E COME TALE VA TRATTATO!

```
printf("STATE");
printf("IMPARANDO");
printf("UN");
printf("SACCO");
printf("DE");
printf("ROBBA!");
```

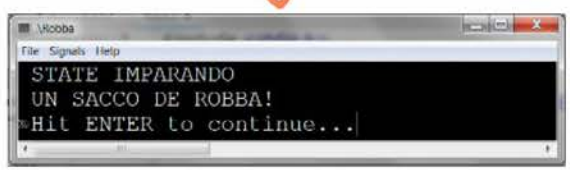
⇒

LE ISTRUZIONI printf SUCCESSIVE ALLA PRIMA SCRIVONO A PARTIRE DALLA POSIZIONE NELL'OUTPUT IMMEDIATAMENTE SUCCESSIVA A QUELLA DELL'ULTIMO CARATTERE SCRITTO DALL'ISTRUZIONE printf PRECEDENTE.

```
printf("STATE IMPARANDO \nUN SACCO DE ROBBA!");
```

⇓

PER ANDARE A CAPO, SI UTILIZZA IL CARATTERE NEWLINE \n



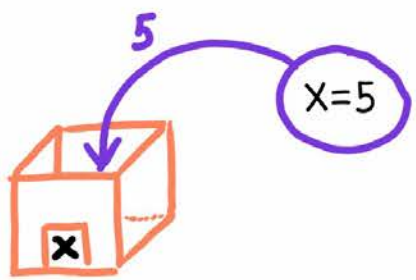
VARIABILE

IN MATEMATICA, UNA VARIABILE È UN CARATTERE ALFABETICO CHE RAPPRESENTA UN NUMERO ARBITRARIO, SCONOSCIUTO, O NON COMPLETAMENTE SPECIFICATO.

IN INFORMATICA, UNA VARIABILE È UNA PORZIONE DI MEMORIA DESTINATA A CONTENERE DEI DATI CHE POTRANNO ESSERE ACCEDUTI O MODIFICATI DURANTE L'ESECUZIONE DI UN PROGRAMMA.

IN MANIERA PIÙ SEMPLICE:

IN INFORMATICA, UNA VARIABILE È UN CONTENITORE DI VALORI



Piu` formalmente …

Concetto di VARIABILE

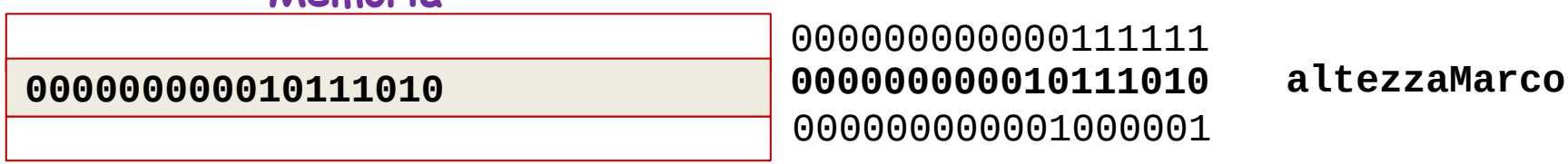
Una variabile, in un programma, è contemporaneamente

- Un **IDENTIFICATORE** …

- Una **LOCAZIONE** di memoria
  contraddistinta da un **INDIRIZZO**!

- Un **VALORE**

```
int altezzaMarco;          /* identificatore: altezzaMarco
                              locazione riservata in memoria, con un certo
                                        indirizzo (0000000000010111010)
                              valore contenuto nella locazione 186        */
```



Piu' formalmente …

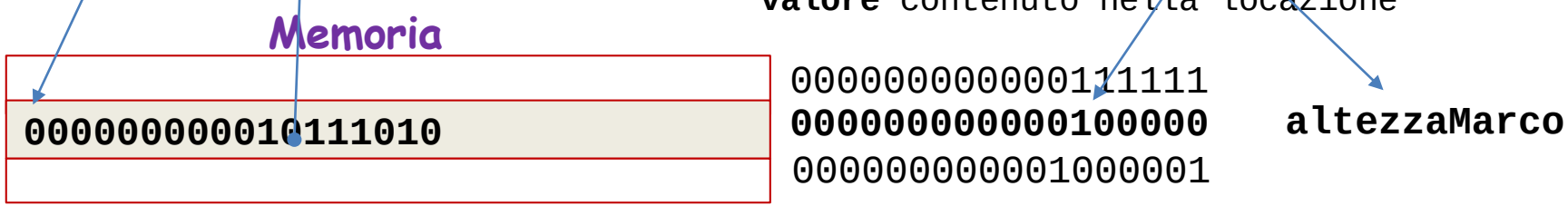
Concetto di VARIABILE

Una variabile, in un programma, è contemporaneamente

- Un **IDENTIFICATORE** cioè il nome della variabile, usato nel programma per … usarla
- Una **LOCAZIONE** di memoria cioè l'area della RAM, riservata per quella variabile, in cui si memorizzano / accedono i valori associati alla variabile (i valori contenuti nella variabile). Questa è contraddistinta da un **INDIRIZZO**!
- Un **VALORE** il valore contenuto nella locazione associata alla variabile

```
int altezzaMarco;
```

```
/* identificatore: altezzaMarco
   locazione riservata in memoria, con un certo
             indirizzo
   valore contenuto nella locazione              */
```



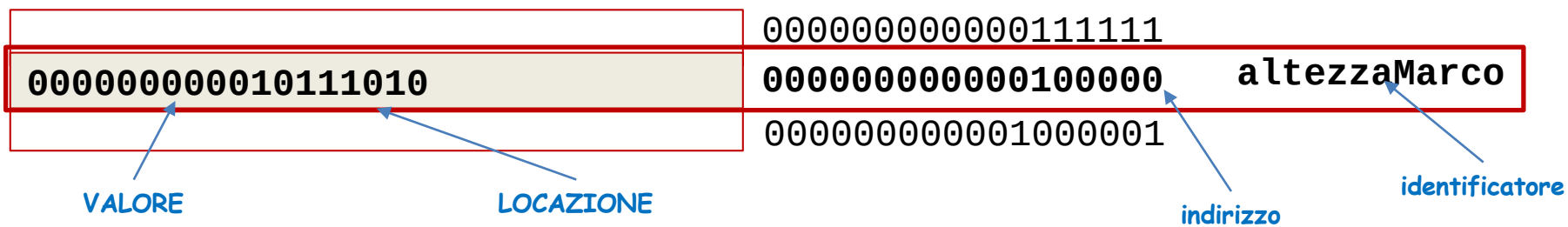
Piu` formalmente …

Concetto di VARIABILE

Una variabile, in un programma, è contemporaneamente

- Un **IDENTIFICATORE** cioè il nome della variabile, usato nel programma per … usarla
- Una **LOCAZIONE** di memoria cioè l'area della RAM, riservata per quella variabile, in cui si memorizzano / accedono i valori associati alla variabile (i valori contenuti nella variabile). Questa è contraddistinta da un **INDIRIZZO**!
- Un **VALORE** il valore contenuto nella locazione associata alla variabile

```
int altezzaMarco;

/* dichiarazione di una variabile denominata altezzaMarco; quando inizierà
l'esecuzione del programma, verrà riservata in memoria una locazione, capace di
contenere un intero rappresentato in forma binaria (complemento a 2); questa
locazione avrà un certo indirizzo. Quando si vuole memorizzare il valore 186 nella
variabile (assegnazione), si memorizza 186 nella locazione. Quando si vuole usare il
contenuto della variabile, ad esempio per stamparlo in OUTPUT, si accede alla
locazione e si usa il valore lì contenuto. */
```



Piu' formalmente …

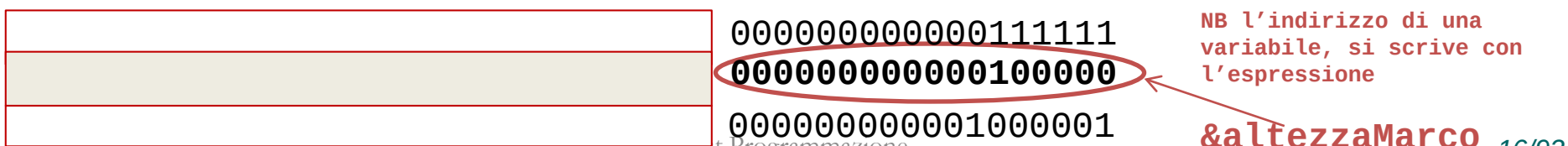
Concetto di VARIABILE

Una variabile, in programma, è contemporaneamente

- Un **IDENTIFICATORE** cioè il nome della variabile, usato nel programma per … usarla)
- Una **LOCAZIONE** di memoria cioè l'area della RAM, riservata per quella variabile, in cui si memorizzano / accedono i valori associati alla variabile (i valori contenuti nella variabile). Questa è contraddistinta da un INDIRIZZO!
- Un **VALORE** il valore contenuto nella locazione associata alla variabile

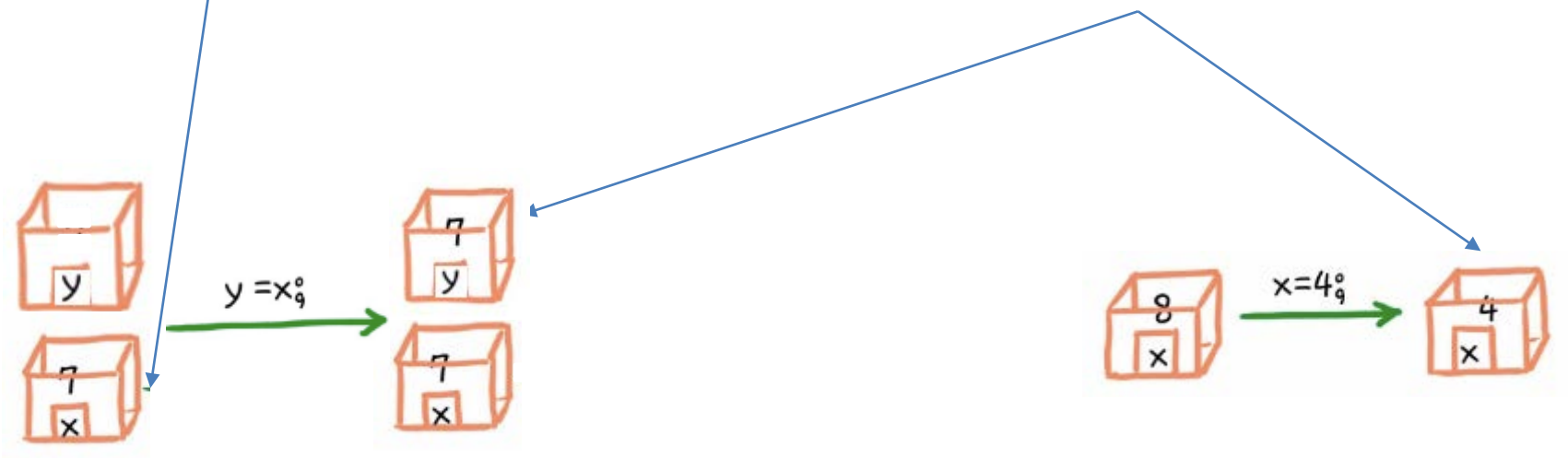
```
int altezzaMarco;                    /* dichiarazione di una variabile chiamata
altezzaMarco; a                                                          apace
di contenere un
locazione avra'                                                          ella
variabile (asse                                                          edere
il valore conte                                                          ne. */
```

se si ha una variabile altezzaMarco, l'espressione per disporre dell'indirizzo della locazione associata fa uso dell'operatore di indirizzamento &



E che ci faccio con una VARIABILE?

Visto che una variabile rappresenta nient'altro che una locazione della RAM
… ci si accede (per vedere che valore c'è) o ci si memorizza un valore …

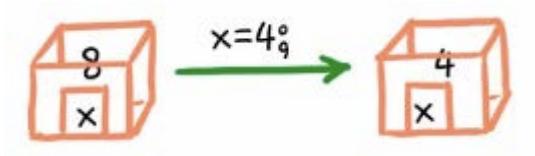


E che ci faccio con una VARIABILE?

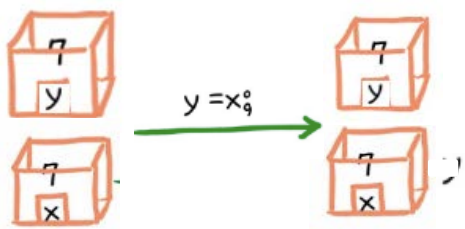
Visto che una variabile rappresenta nient'altro che una locazione della RAM … ci si accede (per vedere che valore c'è) o ci si memorizza un valore …

NB

Se si memorizza qualcosa in una variabile
(cioè si ASSEGNA un VALORE ad una VARIABILE)
il valore precedentemente contenuto nella variabile … non c'è più (c'è quello che abbiamo assegnato or ora …)



Se si accede al VALORE di una VARIABILE (ad esempio per assegnarlo ad un'altra variabile), dopo l'accesso il valore sta ancora nella variabile … gli accessi non sono «distruttivi»



NOME o IDENTIFICATORE

LE VARIABILI HANNO UN **NOME** CHE PERMETTE DI **IDENTIFICARLE**.

- OGNI VARIABILE DEVE AVERE UN NOME DIVERSO

- IL NOME DOVREBBE ESSERE ESPLICATIVO DELLO SCOPO DELLA VARIABILE
  ES: discriminante È UN NOME MIGLIORE DI aX23Dd PER UNA VARIABILE CHE È DESTINATA A MEMORIZZARE IL DISCRIMINANTE DI UN'EQUAZIONE DI 2° GRADO

- UN NOME È UNA SEQUENZA DI CARATTERI ALFANUMERICI, IL CUI PRIMO CARATTERE DEVE ESSERE ALFABETICO (OPPURE UN UNDERSCORE _) E IN GENERE È PREFERIBILMENTE ALFABETICO MINUSCOLO. OGNI PAROLA CHE COSTITUISCE IL NOME, DOPO LA PRIMA, DOVREBBE INIZIARE CON UNA MAIUSCOLA
  ES: radiceReale1, primoNumeroDellaSequenza, minimoCorrente, sequenza3Numeri

camel notation

le variabili hanno un tipo

-ANALISI: QUALI dati? FATTI come (struttura)?

- **INPUT** numeri reali (double)
- **OUTPUT** numero reale
- **dati per calcoli intermedi** ...
- idea: **prod ← b*h**
  area ← prod/2

SINTESI ... =

1) prendere da INPUT (**LETTURA**) i valori da associare a b e h

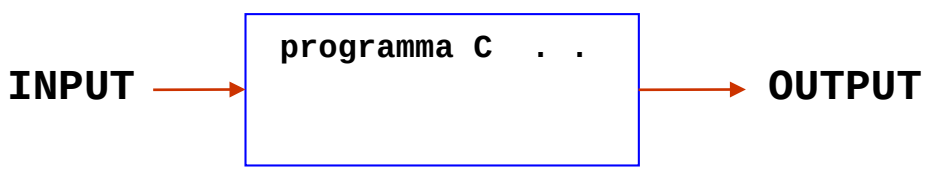
2) calcolare prod (**ASSEGNAZIONE**)

3) calcolare area (**ASSEGNAZIONE**)

4) fornire in OUTPUT (**SCRITTURA**) il valore di area

PROGRAMMAZIONE ... =

```
#include<stdio.h>
int main() {
  double b, h;
  double area;
  double prod;
 scanf("%lf %lf", &b, &h);   1
 prod = b*h;   2
 area = prod/2;   3
 printf("il valore dell'area di un
        triangolo avente base = %g e
        altezza = %g e' %g\n", b, h,
        area);   4
return 0;
}
```



Tipo

LE VARIABILI HANNO UN TIPO CHE RAPPRESENTA L'INSIEME DEI VALORI CHE LA VARIABILE PUÒ ASSUMERE E L'INSIEME DI OPERAZIONI PERMESSE SU QUEI VALORI.

Alcuni tipi

`int` un sottoinsieme dei numeri interi (tutti quelli rappresentabili in forma binaria, in complemento a 2, in una certa quantità di memoria (stabilita di solito dall'ambiente di programmazione … di solito 32 bit …)

`float` un sottoinsieme dei numeri reali (tutti quelli rappresentabili in forma binaria, in Floating Point, in una certa quantità di memoria … es. non meno di 32 bit …)

`double` un sottoinsieme dei numeri reali (tutti quelli rappresentabili in forma binaria, in Floating Point, in una certa quantità di memoria … >= float … per noi 64 bit …)

`char` **`un sottoinsieme dei`** caratteri alfanumerici; una locazione dedicata a valori di questo tipo e' di solito di 1 byte

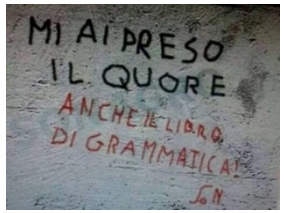
DICHIARAZIONE

PER POTER ESSERE UTILIZZATE, LE VARIABILI **VANNO DICHIARATE** (OPERAZIONE CHE CORRISPONDE A DIRE "UTILIZZERÒ UNA VARIABILE x DI TIPO *int*" OPPURE "UTILIZZERÒ UNA VARIABILE numeroReale DI TIPO *float*").

SINTASSI : *tipo* nome;

ESEMPI : *int* x; *float* radice; *int* y, numero, resto;

SEMANTICA : LA DICHIARAZIONE SERVE A DIRE AL CALCOLATORE : "UTILIZZERÒ UNA VARIABILE home DI TIPO *tipo*, QUINDI PREPARA UN CONTENITORE (UN'AREA DI MEMORIA) PER VALORI DI TIPO *tipo* CON NOME home".



DICHIARAZIONE

LA DICHIARAZIONE DI UNA VARIABILE COMPORTA LA "ALLOCAZIONE" IN MEMORIA CENTRALE DI UNO SPAZIO, DESTINATO A MEMORIZZARE I VALORI ASSUNTI DALLA VARIABILE DURANTE L'ESECUZIONE DEL PROGRAMMA.

LIVELLO LOGICO

LIVELLO FISICO

IL TIPO DI UNA VARIABILE DETERMINA QUALI VALORI LA VARIABILE PUÒ CONTENERE, QUALI OPERAZIONI SI POSSONO EFFETTUARE SULLA VARIABILE E QUANTO SPAZIO VIENE ALLOCATO IN MEMORIA PER LA VARIABILE STESSA.

Istruzione di **Assegnazione**

E' l'operazione che provoca la MEMORIZZAZIONE di un VALORE nella VARIABILE.

Sintassi

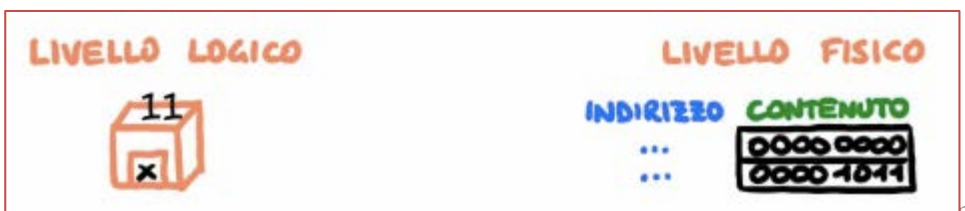
```
nomeVariabile = espressione;
```

Esempi

```
x=11;
```

```
y = (x/5) + 27;
```

Semantica l'espressione viene valutata ed il valore risultante viene memorizzato nella locazione associata alla variabile



Assegnazione

E' l'operazione che provoca la MEMORIZZAZIONE di un VALORE nella VARIABILE.

Sintassi

LEFT VALUE (indirizzo)

RIGHT VALUE (valore)

```
nomeVariabile = espressione
```

una espressione è una scrittura che può combinare operatori ed operandi, oppure rappresentare l'accesso ad una singola variabile; la «esecuzione» di una espressione si chiama «valutazione» e da' luogo ad un valore (di un certo tipo).

Esempi

```
x=11; y=(x/5) + 27;
      radiceReale1=(-b + sqrt(b*b – 4*a*c)) / (2*a)
```

Semantica l'espressione viene valutata ed il valore risultante viene memorizzato nella locazione associata alla variabile

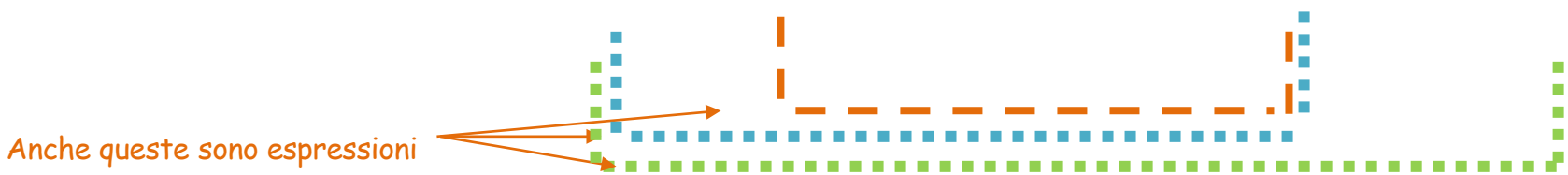
Approfondimento: Espressione!

```
nomeVariabile = espressione;
```

una **espressione** è una scrittura che può combinare operatori ed operandi, oppure rappresentare l'accesso ad una singola variabile; la «esecuzione» di una espressione si chiama «**valutazione**» e da' luogo ad un valore (di un certo tipo).

Esempi

```
x=11;                                   y = (x/5) + 27;
radiceReale1 = (-b + sqrt(b*b - 4*a*c)) / (2*a);
```



ACCESSO

OPERAZIONE CHE PERMETTE DI **ACCEDERE** AL VALORE MEMORIZZATO ALL'INTERNO DI UNA VARIABILE.

SINTASSI : nome — OVVERO L'ACCESSO AD UN VALORE MEMORIZZATO DA UNA VARIABILE SI FA SCRIVENDO IL NOME DELLA VARIABILE

REGOLA : SE nome COMPARE **A SINISTRA** DI UN OPERATORE DI ASSEGNAZIONE LA VARIABILE VIENE USATA **PER MEMORIZZARE** UN VALORE. ALTRIMENTI, LA VARIABILE VIENE USATA **PER ACCEDERE** AL VALORE CHE MEMORIZZA.

SEMANTICA : UTILIZZA IL VALORE MEMORIZZATO IN nome AL POSTO DI nome NELLA PORZIONE DI CODICE IN CUI LA VARIABILE COMPARE.

Esempio: `y=(x/5) + 27;`

viene acceduto il valore di x; questo valore viene diviso per 5, e si ottiene il valore della sotto-espressione (x/5); a questo valore viene aggiunto 27 e si ottiene il risultato della valutazione dell'espressione a destra;

Poi il risultato della valutazione viene memorizzato in y.

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, secondo;    /* i due interi */
  int ris;                     /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu` %d è %d\n",
                                primoNumero, secondo, ris);

return 0;
}
```

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una                                  i viene stampata */
#include <stdio.h>
```

Programma principale

```
int main () {
  int primoNumero, secondo;    /* i due interi */
  int ris;                     /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu' %d è %d\n",
                                primoNumero, secondo, ris);

return 0;
}
```

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, secondo;    /* i due interi */
  int ris;                     /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu' %d è %d\n",
                               primoNumero, secondo, ris);

  return 0;
    }
```

Corpo (body) del programma

{ } racchiudono un **BLOCCO** di istruzioni

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, secondo;   /* i due interi */
                              /* il risultato */

  primoNumero =168;
  secondo = 640;

  int ris;
  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu` %d e` %d\n",
                                primoNumero, secondo, ris);

return 0;
}
```

DICHIARAZIONI DI VARIABILI

/* COMMENTI */
Non sono istruzioni, non verranno "eseguite" … ma sono importanti!!

ourProgram.c

DICHIARAZIONI DI VARIABILI

VANNO MESSE ALL'INIZIO

```
#include <stdio.h>

int main () {
  int primoNumero, secondo;


  primoNumero =168;
  secondo = 640;

  int ris;
  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu` %d e` %d\n",
                                  primoNumero, secondo, ris);

return 0;
}
```

NO

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, secondo;   /* i due interi */
  int ris;                    /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu` %d e` %d\n",
                              primoNumero, secondo, ris);

return 0;
}
```

DICHIARAZIONI DI VARIABILI

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNum                                    eri */
  int ris;                                        ato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;        /* calcolo */

  printf ("il risultato di %d piu` %d e` %d\n",
                                  primoNumero, secondo, ris);

return 0;
}
```

Istruzioni di assegnazione

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, secondo;    /* i due interi */
  int ris;                     /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu` %d e` %d\n",
                                primoNumero, secondo, ris);

return 0;
}
```

Espressione: un operatore e due operandi

Un'espressione combina operatori ed operandi; viene VALUTATA e da' luogo ad un VALORE (il risultato della valutazione)

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, secondo;    /* i due interi */
  int ris;                     /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;      /* calcolo */
```

Pero` anche questa e` un'espressione (640 e` una "costante numerica" inserita nel codice, il cui valore e` ... 640).
640 viene assegnato a secondo.

Un'espressione combina operatori ed operandi;
un **operando** puo` essere

- una variabile (cui si accede): come in ... `ris = primoNumero + secondo;`
- o un valore costante (esempio: 640, 'r', 31.23 ... come in `ris = primoNumero*24;`)
- o un'altra espressione ("sotto-espressione"), da valuare a sua volta (es. `640 + 3*primo`)

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, secondo;    /* i due interi */
  int ris;                     /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = pri                                    */
```

"Istruzione" di stampa

```
  printf ("il risultato di %d piu` %d e` %d\n",
                                 primoNumero, secondo, ris);

return 0;
}
```

Veramente si tratta di una CHIAMATA DI FUNZIONE (una funzione di libreria – la libreria delle funzioni di I/O standard); e' un MODULO, o **sottoprogramma**; il codice di questo sottoprogramma non sta qui, ma nella "libreria oggetto" corrispondente a stdio.h . Quando questa funzione viene chiamata, 1) l'esecuzione del programma principale viene sospesa; 2) viene eseguito il codice del sottoprogramma; 3) l'esecuzione del programma principale riprende.

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, s
  int ris;                                  /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu` %d e` %d\n",
                                  primoNumero, secondo, ris);

return 0;
}
```

DIRETTIVA di inclusione, in questo file, del contenuto di **stdio.h**; questo "file header" contiene le dichiarazioni delle funzioni della libreria di I/O standard, tra le quali printf(). (non il codice di queste funzioni, quello sta nella libreria oggetto

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, secondo;    /* i due interi */
  int ris;                     /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu` %d e` %d\n",
                                primoNumero, secondo, ris);
```

STOP

```
return 0;
}
```

Anche il programma principale e` una funzione … la funzione "main". Quando la sua esecuzione termina, l'struzione return passa all'esterno (a chi ha chiesto l'esecuzione del programma; al Sistema operativo …) un valore … di solito zero

Quale algoritmo rappresenta ourProgram.c ??

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, secondo;    /* i due interi */
  int ris;                     /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu` %d e` %d\n",
                                primoNumero, secondo, ris);

return 0;
}
```

Quale algoritmo rappresenta ourProgram.c ??

0) **Servono primoNumero e secondo e ris (strutture dati …)**
1) **Assegna primoNumero con 168**
2) **Assegna 640 a secondo**
3) **Assegna a ris il risultato di primoNumero+secondo**
4) **Stampa ris**
5) **Fine**

Come sarebbe il programma corrispondente in linguaggio macchina?

Quale preferiamo scrivere?

Quello in C, esatto.

Perché?

Quale algoritmo rappresenta ourProgram.c ??

0) **Servono primoNumero e secondo e ris (strutture dati …)**
1) **Assegna primoNumero con 168**
2) **Assegna 640 a secondo**
3) **Assegna a ris il risultato di primoNumero+secondo**
4) **Stampa ris**
5) **Fine**

Come sarebbe il programma corrispondente in linguaggio macchina?

Quale preferiamo scrivere?

Quello in C, esatto.

Perché?

Perché scrivere un programma direttamente in LM è difficile, poco naturale per un umano, costringe a gestire direttamente la memoria, … già ci basta il C

Pausa di riflessione, e discussione su come effettivamente un programma in C diventa un programma eseguibile, in esecuzione nel computer

Pregi di un LINGUAGGIO AD ALTO LIVELLO

IL C è un linguaggio ad alto livello, termine contrapposto a «basso livello» cioè quello della macchina.

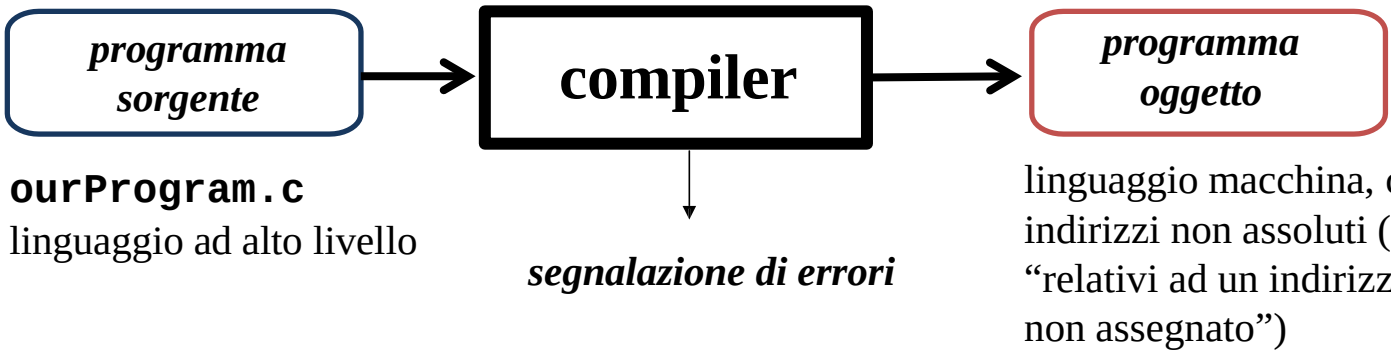
Un programma scritto in LINGUAGGIO AD ALTO LIVELLO,

- è più simile al linguaggio naturale e al modo in cui pensiamo ai problemi … ci permette di usare espressioni e istruzioni piu' simili alle notazioni (matematiche) che usiamo di solito; usiamo parole (le PAROLE CHIAVE del linguaggio) che ci sono vicine e comprensibili (if, while, for, repeat, until, do … non tutte del C) … o comunque più vicine a noi di quanto non sia `0001000` !!

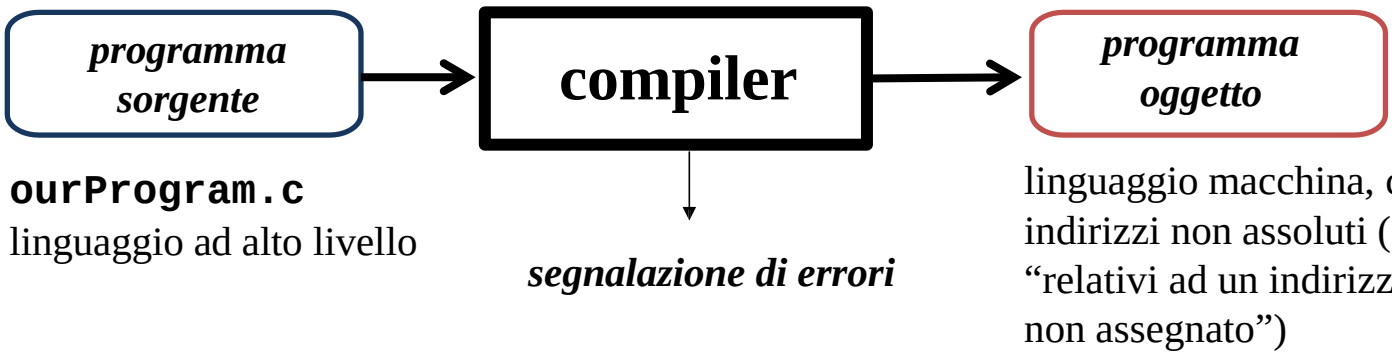
- Portabile (lo stesso programma funziona su qualsiasi macchina, mentre un programma in LM funziona con le CPU che usano quel LM). Come questo succeda si vede tra poco.

Si', CPU diverse hanno linguaggi diversi …

Dal programma in C al programma eseguito – 1/3



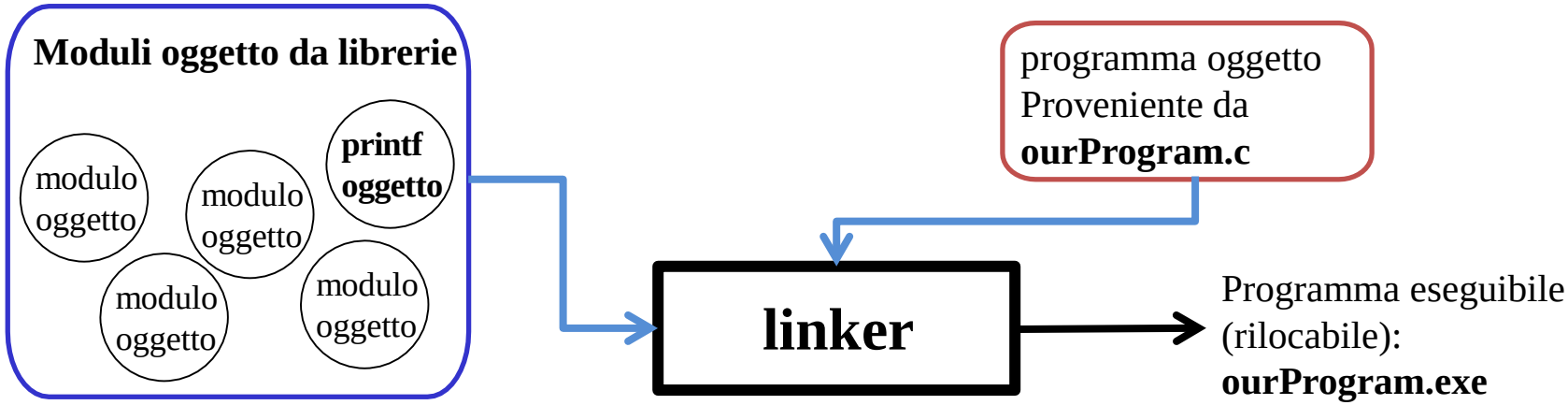
Dal programma in C al programma eseguito – 1/3



Nel programma ci sono inclusioni di librerie (cioè di insiemi di funzioni, definite da altri ma che ci fanno comodo e che usiamo nel programma («chiamando» quelle funzioni).
O anche riferimenti a funzioni sviluppate da noi, e che noi vogliamo usare nel programma.

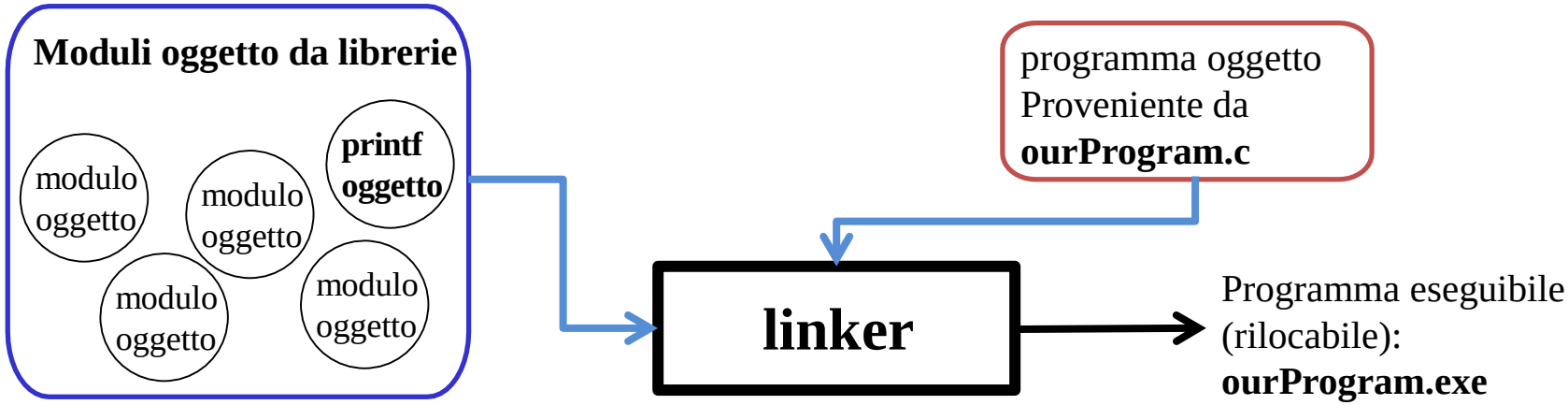
Le funzioni che noi definiamo nel programma (nel file .c) e quelle definite da noi o da altri nelle librerie incluse nel programma, sono anche chiamate «**moduli**» del programma.
I moduli che abbiamo scritto nel file (più avanti parleremo di questo moduli: le funzioni) hanno codice sorgente che verrà compilato.
I moduli esterni vanno compilati a parte ed uniti poi al risultato della compilazione del nostro programma.

Dal programma in C al programma eseguito - 2/3



Run ourProgram

Dal programma in C al programma eseguito - 2/3



Per ottenere un programma eseguibile bisogna prima unire tutti i moduli oggetto: quello del nostro programma, ottenuto con la sua compilazione (andata bene …), e quelli delle funzioni esterne, cioè quelle non scritte da noi nel file ourProgram.c.

Questa unione si chiama *collegamento* (linking) e viene eseguita quando tutti i moduli sono stati compilati e quindi hanno un codice oggetto disponibile.

Per il nostro programma, la compilazione dobbiamo attivarla noi; invece per le funzioni definite nelle librerie la compilazione è automatica e viene curata dall'ambiente di programmazione in cui lavoriamo (come il DEV).

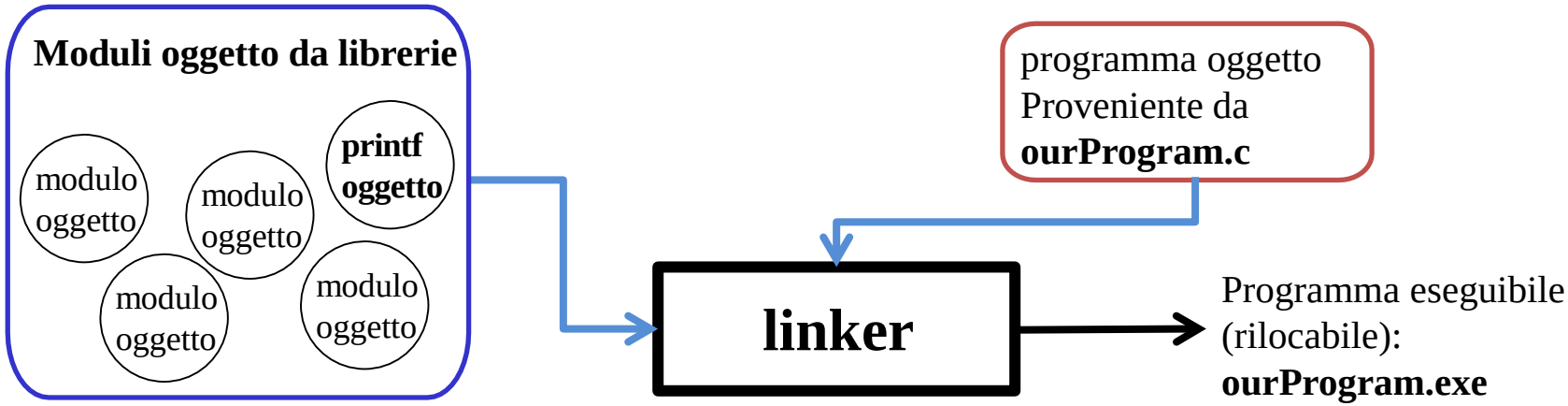
Il risultato del linking è il programma «rilocabile», cioè in sostanza il programma completo di tutte le sue parti, in linguaggio macchina, ma con gli indirizzi che sono tutti definiti rispetto ad un indirizzo base (non ancora noto: sarà noto solo quando effettivamente piazzeremo il programma nella memoria – "caricamento", "loading" - quindi per ora si assume zero).

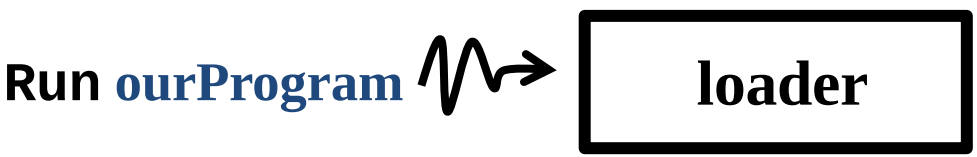
Ora il programma può essere eseguito.

L'indirizzo base viene definito quando il programma viene caricato in memoria centrele (a partire proprio da quel certo indirizzo), cioe' al "Run"

Run ourProgram

Dal programma in C al programma eseguito – 3/3





Il programma viene caricato in memoria centrale, a partire da un indirizzo (occupando una zona libera). Ora tutti gli indirizzi usati nel programma oggetto diventano assoluti ed il programma in linguaggio macchina viene eseguito (vedi la parte di architettura, e in particolar modo il processore.

ourProgram algorithm … without input …

0) Servono primoNumero e secondo e ris (struttura dati …)
1) Assegna primoNumero con 168
2) Assegna 640 a secondo
3) Assegna a ris il risultato di primoNumero+secondo
4) Stampa ris
5) Fine

… and with input

Versione con INPUT ???

ourProgram algorithm … without input …

0) Servono primoNumero e secondo e ris (struttura dati …)
1) Assegna primoNumero con 168
2) Assegna 164 a secondo
3) Assegna a ris il risultato di primoNumero+secondo
4) Stampa ris
5) Fine

… and with input

Versione con INPUT ???
0) (primoNum, secondoNum, ris … i dati che usiamo
1) INPUT primoNum, secondoNum
2) ris = primoNum + secondoNum
3) OUTPUT ris
4) Fine

→ Vogliamo fare il programma ?

ourProgram.c with input

```
/* programma che esegue la somma di due valori interi letti da imput. I
valori sono letti in due variabili intere. La loro somma viene assegnata
ad una terza var, che poi viene stampata */

#include <stdio.h>
int main () {
  int primoNumero, secondoNumero,    /* i dati da input */
      ris;                           /* risultato della somma */




return 0;
}
```

passo 0)

ourProgram.c with input

```
/* programma che esegue la somma di due valori interi letti da imput. I
valori sono letti in due variabili intere. La loro somma viene assegnata
ad una terza var, che poi viene stampata */

#include <stdio.h>
int main () {
  int primoNumero, secondoNumero,    /* i dati da input */
      ris;                           /* risultato della somma */

  printf ("Oh utente, scrivi due numeri e io li sommo: ");
  scanf("%d %d", &primoNumero, &secondoNumero);




return 0;
}
```

passo 1) (chiedere per avere e` intelligente …)

ourProgram.c with input

```
/* programma che esegue la somma di due valori interi letti da imput. I
valori sono letti in due variabili intere. La loro somma viene assegnata
ad una terza var, che poi viene stampata */

#include <stdio.h>
int main () {
  int primoNumero, secondoNumero,    /* i dati da input */
      ris;                           /* risultato della somma */

  printf ("Oh utente, scrivi due numeri e io li sommo: ");
  scanf("%d %d", &primoNumero, &secondoNumero);

  ris = primoNumero + secondoNumero;          /* calcolo */



return 0;
}
```

passo 2)

ourProgram.c with input

```
/* programma che esegue la somma di due valori interi letti da imput. I
valori sono letti in due variabili intere. La loro somma viene assegnata
ad una terza var, che poi viene stampata */

#include <stdio.h>
int main () {
  int primoNumero, secondoNumero,    /* i dati da input */
      ris;                           /* risultato della somma */

  printf ("Oh utente, scrivi due numeri e io li sommo: ");
  scanf("%d %d", &primoNumero, &secondoNumero);

  ris = primoNumero + secondoNumero;          /* calcolo */

  printf (" Stimato user, %d piu` %d e` uguale a %d\n",
                        primoNumero, secondoNumero, ris);

return 0;
}
```

passo 3)

passo 4)

Vedi ora Approfondimenti PRIMA PARTE

Tecniche della Programmazione, lez. 3

Algoritmi e diagrammi di flusso

Flusso?

Non ci piace il salto

Verso la programmazione Strutturata

Algoritmo (again …)

Abbiamo detto che e` una sequenza di PASSI, operazioni, istruzioni …
per risolvere un problema per il quale abbiamo una formalizzazione matematica

Più schematicamente ci sono delle componenti da considerare

- **INFORMAZIONI** relative al problema: rappresentate come **DATI** (nel calcolatore, nell'algoritmo, nel programma)
- **INPUT** … (cosa sono? Vedi approfondimenti)
- **Procedura Computazionale** passi di calcolo … (quali sono? Vedi appr.)
- **OUTPUT** … (? vedi …)

Progettazione Algoritmo: attivita` creativa …
Esecuzione Algoritmo: attivita` meccanica

"Esecuzione dell'algoritmo" (*Flusso di esecuzione*)

0) primoNumero, secondo, ris …
1) **Assegna primoNumero con 68**
2) **Assegna 64 a secondo**
3) **Assegna a ris il risultato di primoNumero+secondo**
4) **Stampa ris**
5) **Fine**

tutte le esecuzioni sono uguali … i risultati non cambiano da istanza ad istanza ... e nemmeno il flusso

Flusso di esecuzione: 1) 2) 3) 4) 5)

Versione con INPUT

0) (primoNum, secondoNum, ris … i dati che usiamo
1) **INPUT primoNum, secondoNum**
2) **ris = primoNum + secondoNum**
3) **OUTPUT ris**
4) **Fine**

esecuzioni diverse, a seconda dell'INPUT, ma comunque la sequenza di passi e` sempre la me**des**ima …

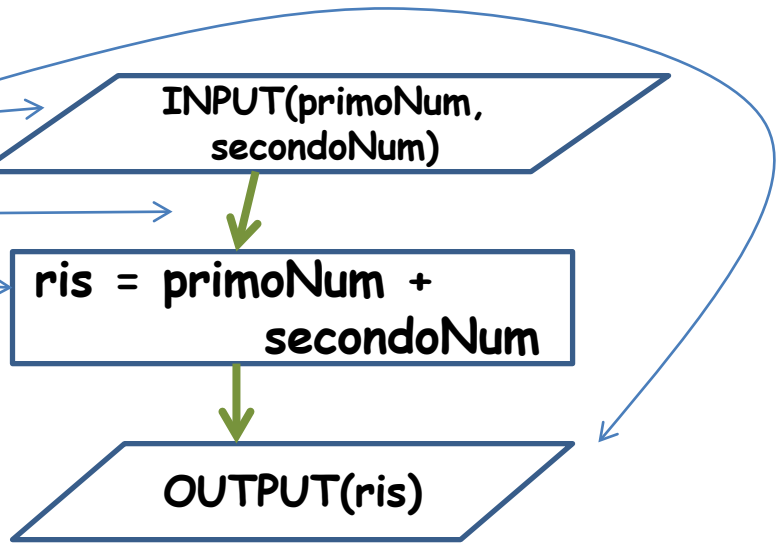
Flusso di esecuzione: 1) 2) 3) 4)

Diagrammi di Flusso

Anche Diagrammi a Blocchi.
Un formalismo grafico per rappresentare algoritmi
(e seguire il flusso di esecuzione)

Istruzioni di INPUT/ OUTPUT
Direzione del Flusso di esecuzione
Istruzione generica

1) INPUT primoNum, secondoNum
2) ris = primoNum + secondoNum
3) OUTPUT ris
4) Fine



Flusso di esecuzione: 1) → 2) → 3) → 4)

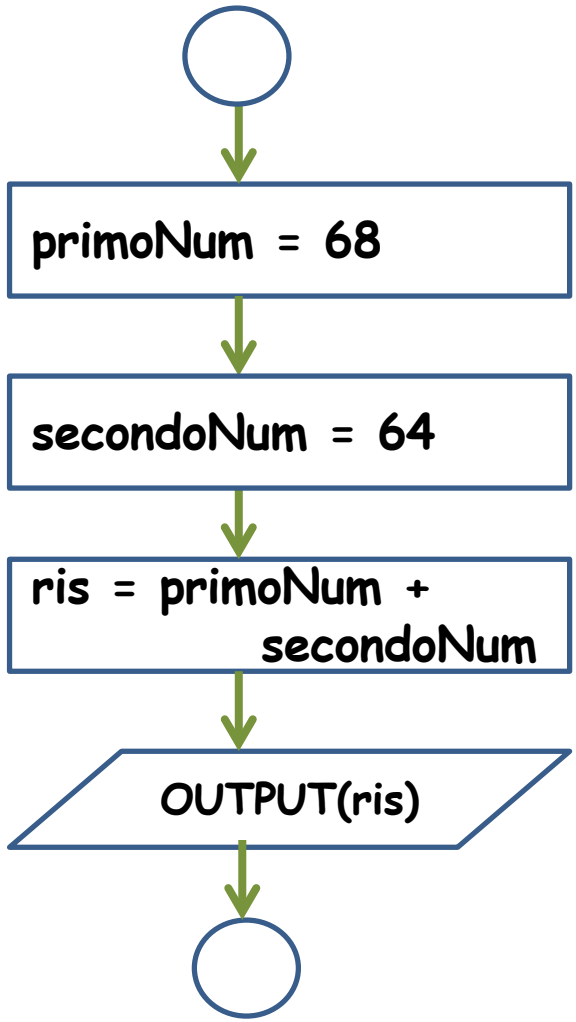
4) ? → STOP?

Diagrammi di Flusso

Anche *Diagrammi a Blocchi*. Un formalismo grafico per rappresentare algoritmi (e seguire il flusso di esecuzione)

1) Assegna primoNum con 68
2) Assegna 64 a secondoNum
3) Assegna a ris il risultato di primoNum+secondoNum
4) Stampa ris
5) Fine

Flusso di esecuzione: 1) → 2) → 3) → 4) → 5)

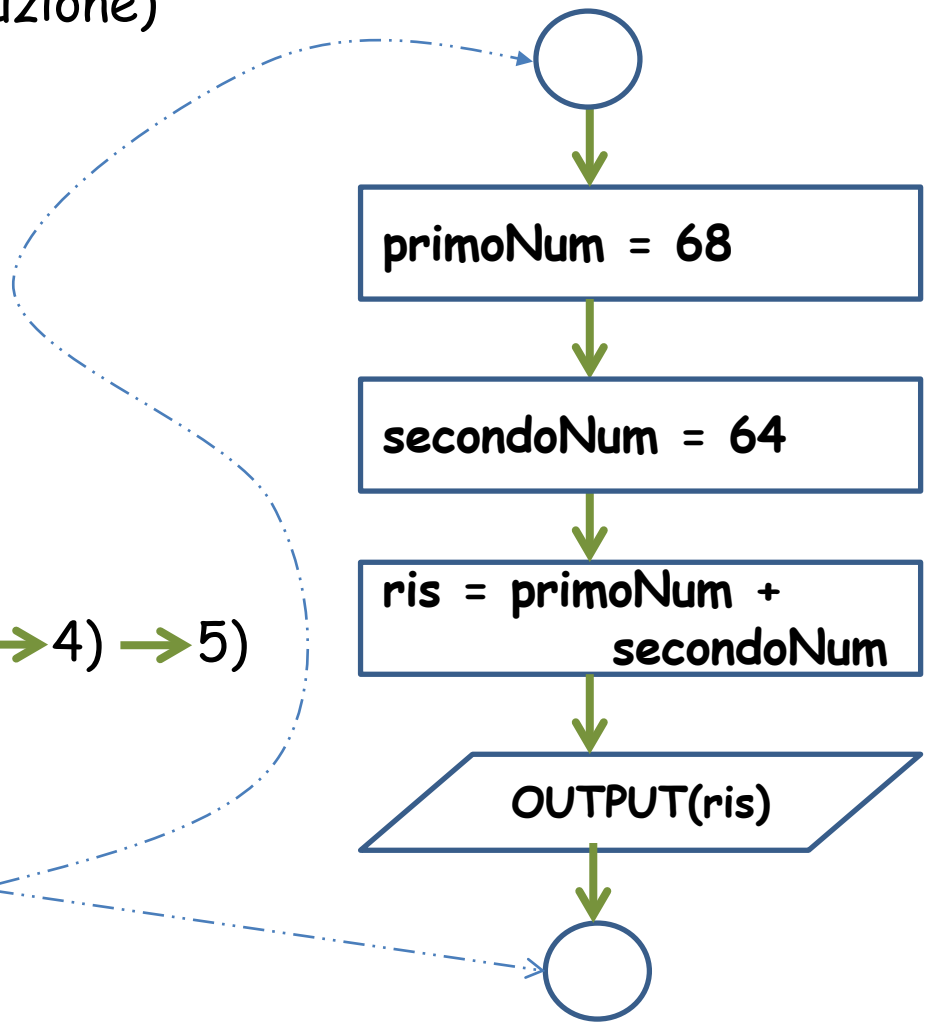


Diagrammi di Flusso

Anche Diagrammi a Blocchi. Un formalismo grafico per rappresentare algoritmi (e seguire il flusso di esecuzione)

1) Assegna primoNum con 68
2) Assegna 64 a secondoNum
3) Assegna a ris il risultato di primoNum+secondoNum
4) Stampa ris
5) Fine

Flusso di esecuzione: 1) → 2) → 3) → 4) → 5)



Questi pallocchi sono un vecchio modo per simboleggiare il punto di entrata e di uscita dal programma

Li ometteremo in seguito

Algoritmo per l'eq. di secondo grado

Dati i coefficienti a,b,c di un'equazione di secondo grado, calcolare le soluzioni reali, supponendo che esistano soluzioni reali.

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaQuadro, … 2*a, 4*a*c …
1) INPUT (a, b, c)
2) ☺
3) x1 = meno b più radice-di-deltaQuadro, tutto diviso 2-per-a
4) x2 = (-b - $\sqrt{deltaQuadro}$) / 2*a
5) ☺

☺ Scrivere qualcosa al posto di ☺

Algoritmo per l'eq. di secondo grado

Dati i coefficienti a,b,c di un'equazione di secondo grado, calcolare le soluzioni reali, supponendo che esistano soluzioni reali.

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaQuadro, … 2*a, 4*a*c …
1) INPUT (a, b, c)
2) deltaQuadro = b*b -4*a*c
3) x1 = formula per la prima soluzione
4) x2 = formula per la seconda soluzione
5) OUTPUT(x1,x2)

Algoritmo per l'eq. di secondo grado

Dati i coefficienti a,b,c di un'equazione di secondo grado, calcolare le soluzioni reali, supponendo che esistano soluzioni reali.

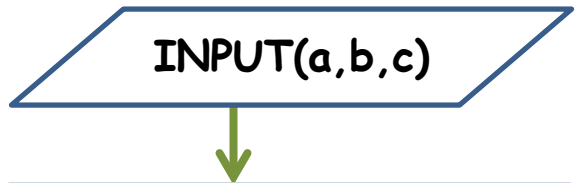
0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaQuadro, … 2*a, 4*a*c …
1) INPUT (a, b, c)
2) deltaQuadro = b*b -4*a*c
3) x1 = formula per la prima soluzione
4) x2 = formula per la seconda soluzione
5) OUTPUT(x1,x2)

☺ Diagramma di flusso? ☺

Algoritmo per l'eq. di secondo grado

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaquadro, … 2*a, 4*a*c …

1) INPUT (a, b, c)
2) deltaQuadro = b*b -4*a*c
3) x1 = formula per la prima soluzione
4) x2 = formula per la seconda soluzione
5) OUTPUT(x1,x2)

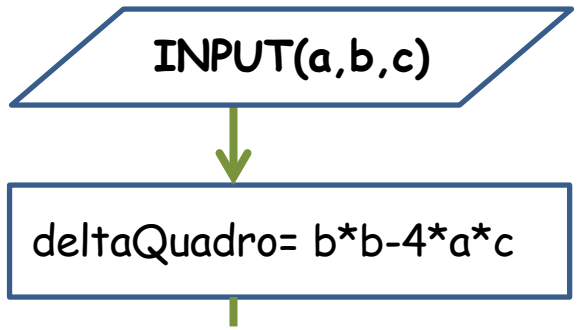



…

Algoritmo per l'eq. di secondo grado

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaquadro, … 2*a, 4*a*c …

1) INPUT (a, b, c)
2) deltaQuadro = b*b -4*a*c
3) x1 = formula per la prima soluzione
4) x2 = formula per la seconda soluzione
5) OUTPUT(x1,x2)

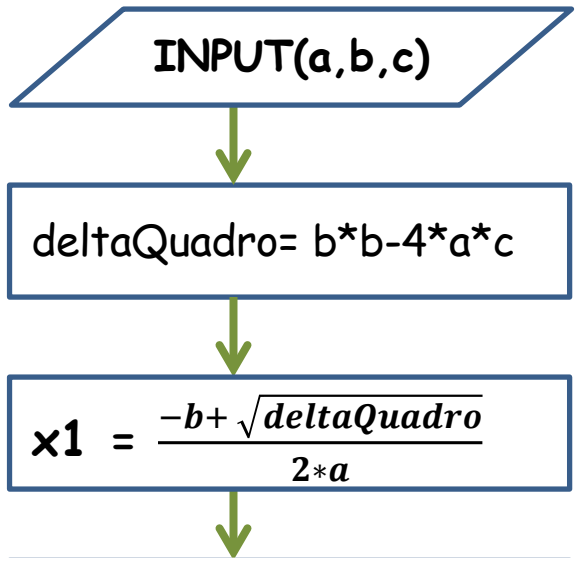


☺…

Algoritmo per l'eq. di secondo grado

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaquadro, … 2*a, 4*a*c …

1) INPUT (a, b, c)
2) deltaQuadro = b*b -4*a*c
3) x1 = formula per la prima soluzione
4) x2 = formula per la seconda soluzione
5) OUTPUT(x1,x2)

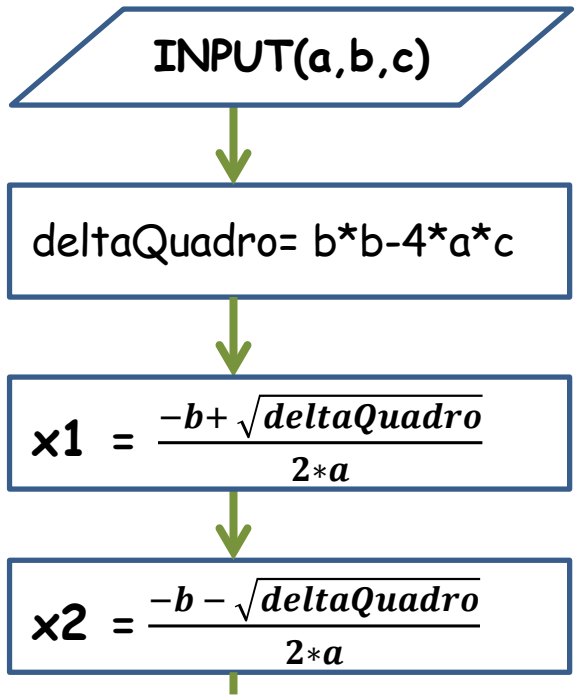


☺…

Algoritmo per l'eq. di secondo grado

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaquadro, … 2*a, 4*a*c …

1) INPUT (a, b, c)
2) deltaQuadro = b*b -4*a*c
3) x1 = formula per la prima soluzione
4) x2 = formula per la seconda soluzione
5) OUTPUT(x1,x2)

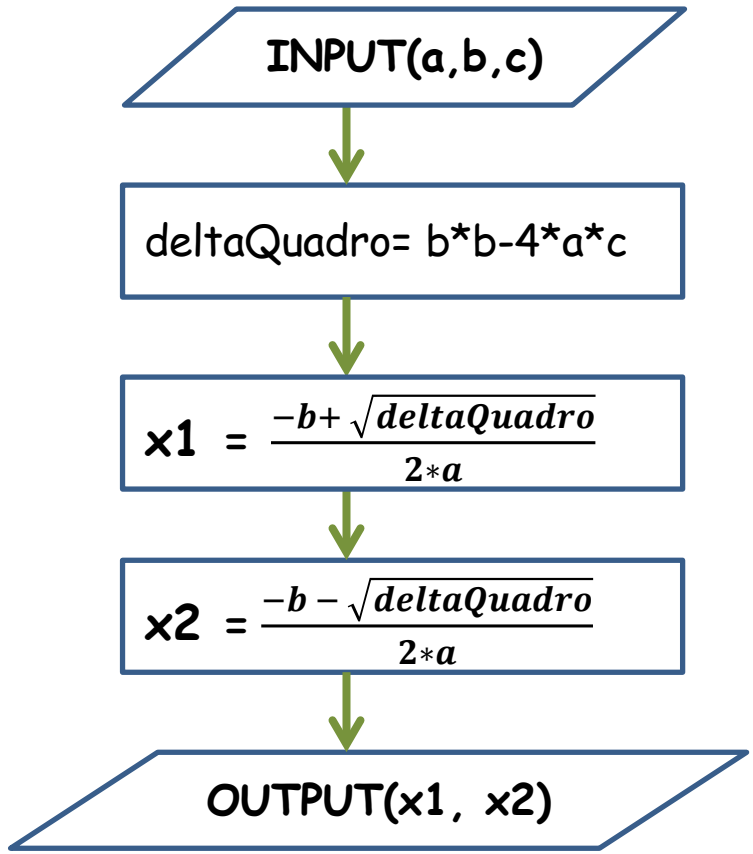


☺…

Algoritmo per l'eq. di secondo grado

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaquadro, … 2*a, 4*a*c …

1) INPUT (a, b, c)
2) deltaQuadro = b*b -4*a*c
3) x1 = formula per la prima soluzione
4) x2 = formula per la seconda soluzione
5) OUTPUT(x1,x2)

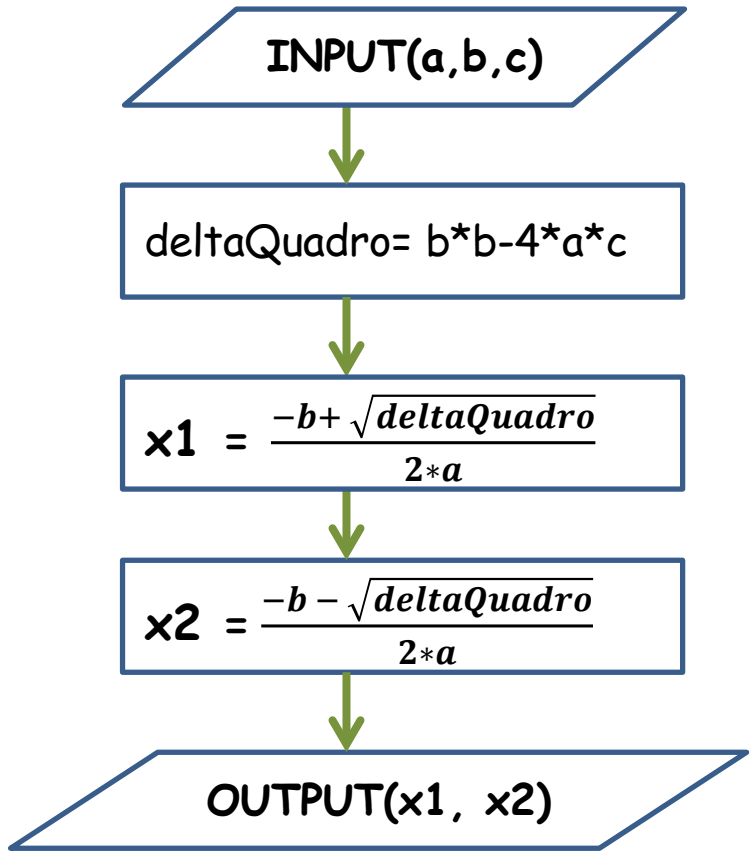


☺ Flusso di esecuzione per a=1, b=4, c=1 ? ☺

Algoritmo per l'eq. di secondo grado

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaquadro, … 2*a, 4*a*c …

1) INPUT (a, b, c)
2) deltaQuadro = b*b -4*a*c
3) x1 = formula per la prima soluzione
4) x2 = formula per la seconda soluzione
5) OUTPUT(x1,x2)



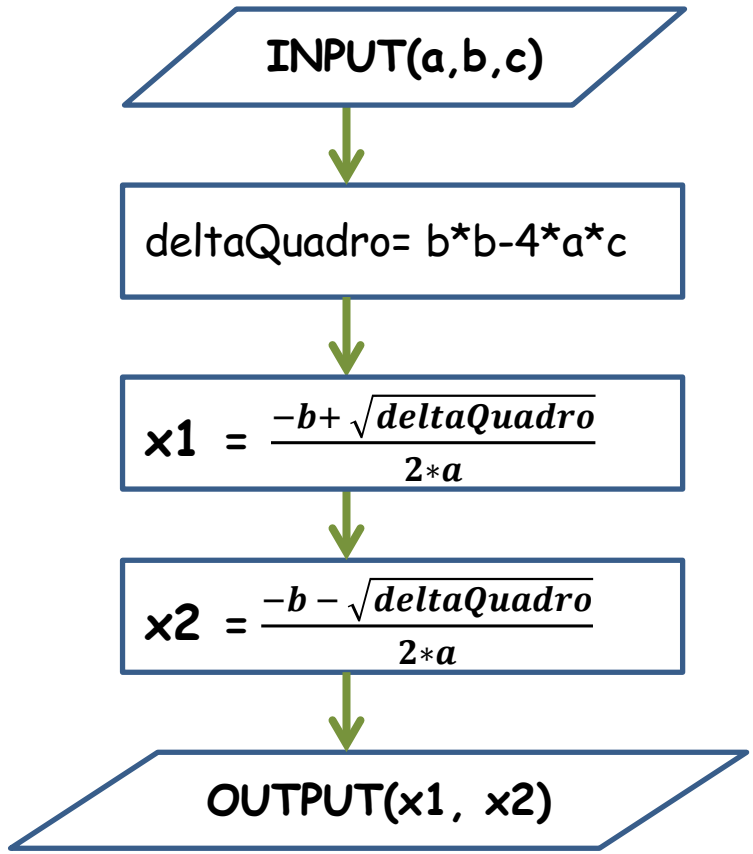
Flusso di esecuzione per a=1, b=4, c=1 :

1) → 2) → 3) → 4) → 5)

Algoritmo per l'eq. di secondo grado

0) i dati: a, b, c; i risultati, x1, x2 (supponiamo che esistano reali); i dati intermedi deltaquadro, … 2*a, 4*a*c …

1) INPUT (a, b, c)
2) deltaQuadro = b*b -4*a*c
3) x1 = formula per la prima soluzione
4) x2 = formula per la seconda soluzione
5) OUTPUT(x1,x2)



Flusso di esecuzione per a=1, b=4, c=1 :
1) → 2) → 3) → 4) → 5)

OBS.
(per qualunque istanza il flusso e` sempre quello … anche se ovviamente input e output possono cambiare)

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore.

BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4) ?

cosa sappiamo di certo su questo numero?

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore.

BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?

Sicuramente è un divisore di 4 (non può essere più grande di 4; o è 4 o è un numero più piccolo, al limite 1)

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore. BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?

- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore. BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente è un divisore di 4 (non può essere più grande di 4; o è 4 o è un numero più piccolo, al limite 1)

- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (in cui calcoliamo il MCD)
1) INPUT (n, m)

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore. BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente è un divisore di 4 (non può essere più grande di 4; o è 4 o è un numero più piccolo, al limite 1)

- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (con cui calcoliamo il MCD)
1) INPUT (n, m) (inizializzazione di n ed m, mediante operazione di input)
2) ris = ☺ (inizializzazione di ris mediante assegnazione diretta)

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore. BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente è un divisore di 4 (non può essere più grande di 4; o è 4 o è un numero più piccolo, al limite 1)

- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (con cui calcoliamo il MCD)
1) INPUT (n, m)
2) ris = ☺

3) SE ris «DIVIDE n» E «DIVIDE m» EUREKA!
   …

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore. BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente e` un divisore di 4 (non puo` essere piu` grande di 4; o e` 4 o e` un numero piu` piccolo, al limite 1)

- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (con cui calcoliamo il MCD)
1) INPUT (n, m)
2) ris = ☺
3) SE ris «DIVIDE n» E «DIVIDE m» EUREKA!
   1) eureka e poi?
   2) E sennò?? Diciamo che "sennò proseguiamo ma con un altro valore per **ris**"

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore. BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente è un divisore di 4 (non può essere più grande di 4; o è 4 o è un numero più piccolo, al limite 1)

- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (con cui calcoliamo il MCD)
1) INPUT (n, m)
2) ris = ☺
3) SE ris «DIVIDE n» E «DIVIDE m» EUREKA!
   eureka e poi?
   E sennò?? Diciamo che "sennò proseguiamo ma con un altro valore per ris"
4) ris = ris -1

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore. BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente e` un divisore di 4 (non puo` essere piu` grande di 4; o e` 4 o e` un numero piu` piccolo, al limite 1)

- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (con cui calcoliamo il MCD)
1) INPUT (n, m)
2) ris = ☺
3) SE ris «DIVIDE n» E «DIVIDE m» EUREKA!
   eureka e poi?
   E sennò?? Diciamo che "sennò proseguiamo ma con un altro valore per ris"
4) ris = ris -1
   ... e poi??

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore.
BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente e` un divisore di 4 (non puo` essere piu` grande di 4; o e` 4 o e` un numero piu` piccolo, al limite 1)

- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (con cui calcoliamo il MCD)
1) INPUT (n, m)
2) ris = ☺
3) SE ris «DIVIDE n» E «DIVIDE m» EUREKA!
   eureka e poi?
   E sennò?? Diciamo che "sennò proseguiamo ma con un altro valore per ris"
4) ris = ris -1
   E poi torna al passo 3

(+/- a questo punto del flusso abbiamo fatto tutti i calcoli e possiamo chiudere mandando in output il MCD)

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore.
BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente e` un divisore di 4 (non puo` essere piu` grande di 4; o e` 4 o e` un numero piu` piccolo, al limite 1)

- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (con cui calcoliamo il MCD)
1) INPUT (n, m)
2) ris = ☺ ☺ ☺ ☺ ☺ ☺ ☺
3) SE ris «DIVIDE n» E «DIVIDE m» EUREKA!
   eureka e poi?
   E sennò?? Diciamo che "sennò proseguiamo ma con un altro valore per ris"
4) ris = ris -1
   E poi torna al passo 3
5) OUTPUT(ris)

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore.
BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente e` un divisore di 4 (non puo` essere piu` grande di 4; o e` 4 o e` un numero piu` piccolo, al limite 1)

- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (con cui calcoliamo il MCD)
1) INPUT (n, m)
2) ris = **minimo tra n ed m**
3) SE ris «DIVIDE n» E «DIVIDE m» EUREKA!
   eureka e poi?
   E sennò?? Diciamo che "sennò proseguiamo ma con un altro valore per ris"
4) ris = ris -1
   E torna al passo 3
5) OUTPUT(ris)

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore.
BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente e` un divisore di 4 (non puo` essere piu` grande di 4; o e` 4 o e` un numero piu` piccolo, al limite 1)
- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (con cui calcoliamo il MCD)
1) INPUT (n, m)
2) ris = minimo{n,m}
3) SE ris «DIVIDE n» E «DIVIDE m» EUREKA!
   eureka e poi?
   E sennò?? Diciamo che "sennò proseguiamo ma con un altro valore per ris
4) ris = ris -1
   E torna al passo 3
5) OUTPUT(ris)

Eureka che?? Vuoi dire "ris è il MCD dei due numeri". Ok, ma poi non dovremmo finire?

Eeeek, Come finisco?

Come ci arrivo al passo 5)?!

E Torna? Torna?!?!

Che vuol dire Torna?

… calma

Algoritmo per il MCD

Dati due numeri interi, n, m, maggiori di zero, calcolare il Massimo Comun Divisore.
BTW, MCD(9,81)=9, MCD(37,7)=1, MCD(6,4)=2;

MCD(6,4)=?
Sicuramente è un divisore di 4 (non puo` essere piu` grande di 4; o e` 4 o e` un numero piu` piccolo, al limite 1)
- 4 divide sia 6 che 4? No
- 3? No
- 2? Si` !!

0) i dati: n, m (input) e ris (con cui calcoliamo il MCD)
1) INPUT (n, m)
2) ris = minimo{n,m}

3) SE ris «DIVIDE n» E «DIVIDE m»
   Vai al passo 5) con l'idea che lì finisce
   Sennò prosegui naturalmente con il passo successivo
4) ris = ris -1
   E torna al passo 3

5) OUTPUT(ris)

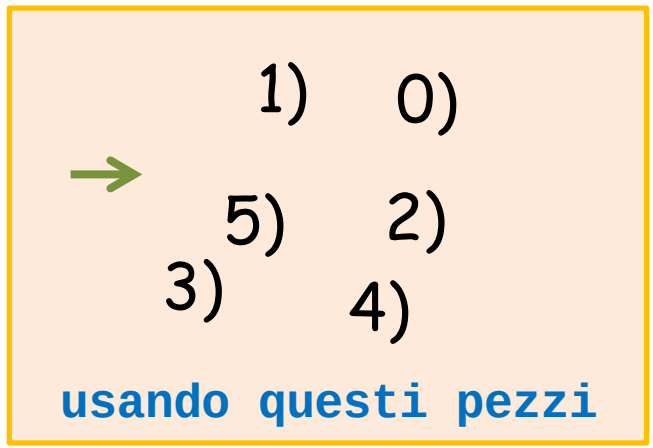
Vai? Torna? Ma il flusso non è sequenziale? Dopo la 3 c'è la 4 o la 5??

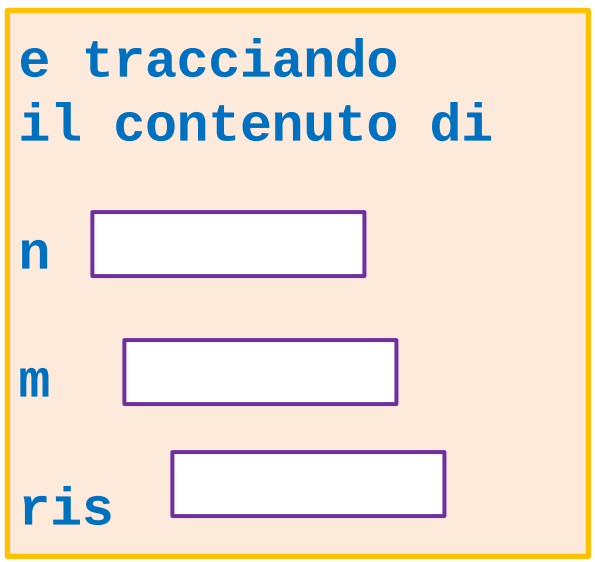
Algoritmo per il MCD (il flusso di esecuzione)

Il flusso di esecuzione può dipendere dal verificarsi di condizioni durante i calcoli …

Flusso di esecuzione per n=14, m=28:

Flusso di esecuzione per n=6, m=4:



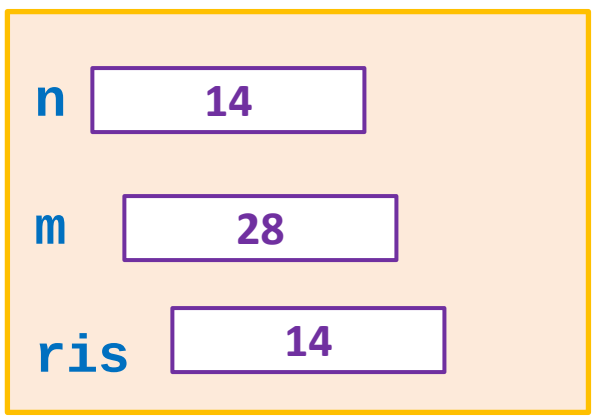


0) i dati: n, m (input) e ris
1) INPUT (n, m)
2) ris = minimo{n,m}
3) SE ris «DIVIDE n» E «DIVIDE m»
   Vai al passo 5) per finire
4) ris = ris -1
   E torna al passo 3
5) OUTPUT(ris)

Algoritmo per il MCD (il flusso di esecuzione)

Il flusso di esecuzione puo` dipendere dal verificarsi di condizioni durante i calcoli …

Flusso di esecuzione per n=14, m=28:
1) → 2) → 3) → 5) e stampa 14



0) i dati: n, m (input) e ris
1) INPUT (n, m)
2) ris = minimo{n,m}
3) SE ris «DIVIDE n» E «DIVIDE m»
Vai al passo 5) per finire
4) ris = ris -1
E torna al passo 3
5) OUTPUT(ris)

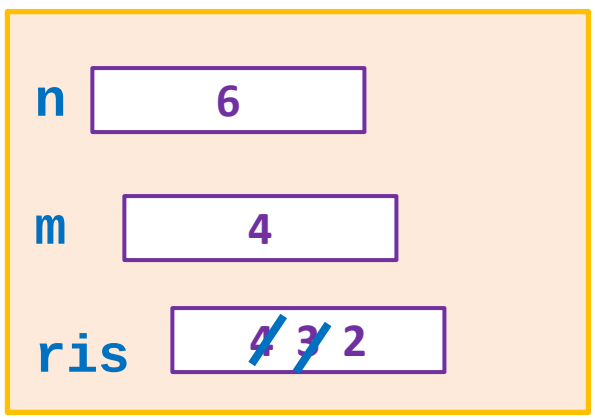
Algoritmo per il MCD (il flusso di esecuzione)

Il flusso di esecuzione può dipendere dal verificarsi di condizioni durante i calcoli …

Flusso di esecuzione per n=6, m=4:

1) → 2) → 3) → 4) → 3) → 4) → 3) → 5)

e stampa 2 perché ris è diventato 2



0) i dati: n, m (input) e ris
1) INPUT (n, m)
2) ris = minimo{n,m}
3) SE ris «DIVIDE n» E «DIVIDE m»
   Vai al passo 5) per finire
4) ris = ris -1
   E torna al passo 3
5) OUTPUT(ris)

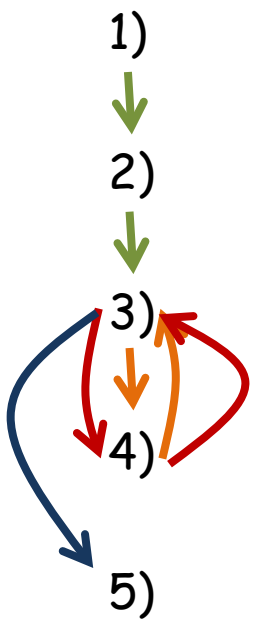
Algoritmo per il MCD (il flusso di esecuzione)

Il flusso di esecuzione può dipendere dal verificarsi di condizioni durante i calcoli …

Flusso di esecuzione per n=14, m=28:
1) → 2) → 3) → 5)

Flusso di esecuzione per n=6, m=4:
1) → 2) → 3) → 4) → 3) → 4) → 3) → 5)

```
0) i dati: n, m (input) e ris
1)    INPUT (n, m)
2)    ris = minimo{n,m}
3)    SE ris «DIVIDE n» E «DIVIDE m»
            Vai al passo 5) per finire
4)    ris = ris -1
            E torna al passo 3
5)    OUTPUT(x1,x2)
```



NB
Algoritmo = sequenza di passi progettata per risolvere un problema

Flusso di esecuzione = sequenza dei passi effettivamente eseguiti durante l'esecuzione dell'algoritmo su un'istanza del problema

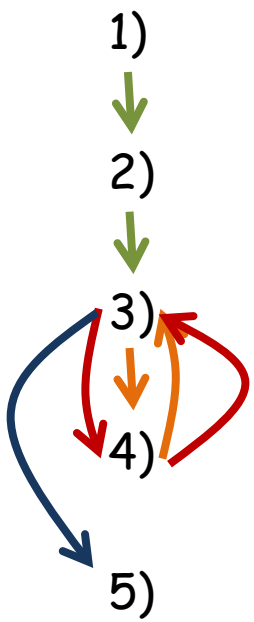
Algoritmo per il MCD (il flusso di esecuzione)

Il flusso di esecuzione può dipendere dal verificarsi di condizioni durante i calcoli …

Flusso di esecuzione per n=14, m=28:
1) → 2) → 3) → 5)

Flusso di esecuzione per n=6, m=4:
1) → 2) → 3) → 4) → 3) → 4) → 3) → 5)

```
0) i dati: n, m (input) e ris
1)    INPUT (n, m)
2)    ris = minimo{n,m}
3)    SE ris «DIVIDE n» E «DIVIDE m»
            Vai al passo 5) per finire
4)    ris = ris -1
            E torna al passo 3
5)    OUTPUT(x1,x2)
```



NB
Algoritmo = sequenza di passi progettata per risolvere un problema

Flusso di esecuzione = sequenza dei passi effettivamente eseguiti durante l'esecuzione dell'algoritmo su un'istanza del problema

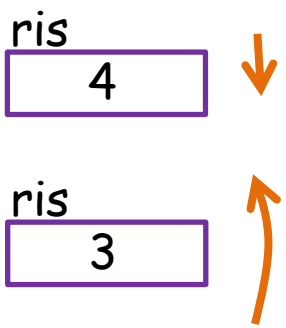
Algoritmo per il MCD (il flusso di esecuzione)

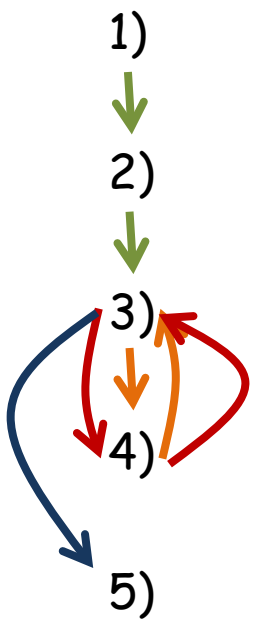
Il flusso di esecuzione può dipendere dal verificarsi di condizioni durante i calcoli …

Flusso di esecuzione per n=14, m=28:
1) → 2) → 3) → 5)

Flusso di esecuzione per n=6, m=4:
1) → 2) → 3) → 4) → 3) → 4) → 3) → 5)

```
0) i dati: n, m (input) e ris
1)    INPUT (n, m)
2)    ris = minimo{n,m}
3)    SE ris «DIVIDE n» E «DIVIDE m»
            Vai al passo 5) per finire
4)    ris = ris -1
            E torna al passo 3
5)    OUTPUT(x1,x2)
```





NB
Algoritmo = sequenza di passi progettata per risolvere un problema

Flusso di esecuzione = sequenza dei passi effettivamente eseguiti durante l'esecuzione dell'algoritmo su un'istanza del problema

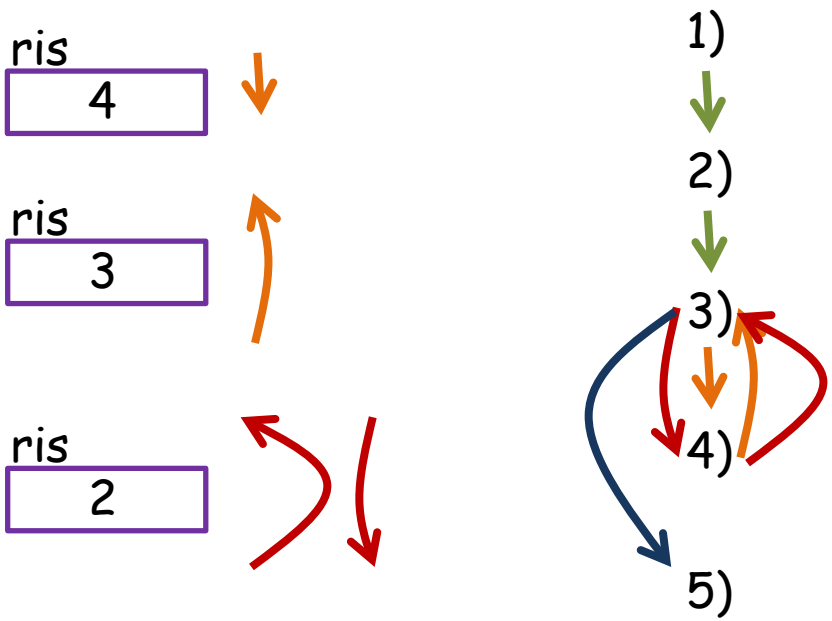
Algoritmo per il MCD (il flusso di esecuzione)

Il flusso di esecuzione può dipendere dal verificarsi di condizioni durante i calcoli …

Flusso di esecuzione per n=14, m=28:
1) → 2) → 3) → 5)

Flusso di esecuzione per n=6, m=4:
1) → 2) → 3) → 4) → 3) → 4) → 3) → 5)

```
0) i dati: n, m (input) e ris
1)    INPUT (n, m)
2)    ris = minimo{n,m}
3)    SE ris «DIVIDE n» E «DIVIDE m»
            Vai al passo 5) per finire
4)    ris = ris -1
            E torna al passo 3
5)    OUTPUT(x1,x2)
```



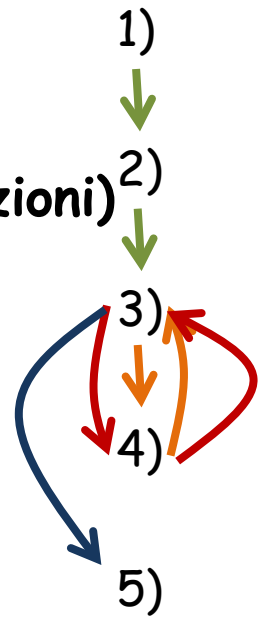
NB
Algoritmo = sequenza di passi progettata per risolvere un problema

Flusso di esecuzione = sequenza dei passi effettivamente eseguiti durante l'esecuzione dell'algoritmo su un'istanza del problema

Ok, il flusso di esecuzione e` …

il risultato del **controllo** cui viene sottoposta l'esecuzione di un algoritmo

ALGORITMO = SEQUENZA DI PASSI CONTROLLATI



Il controllo consiste nel determinare la sequenza di passi (istruzioni) eseguiti/e

1) Di norma: un passo dopo l'altro, nell'ordine dei passi
(la prossima istruzione da eseguire è quella immediatamente successiva a quella in esecuzione)

2) Metodo Vintage: si usa un'istruzione di SALTO
(la prossima istruzione da eseguire è indicata dall'istruzione di salto).

3) Programmazione Strutturata 🥰
(si usano specifiche "***istruzioni di controllo***")

Vedi ora
Approfondimenti
SECONDA PARTE

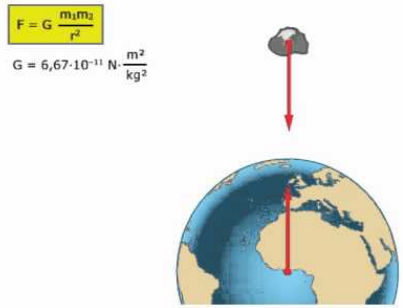
Tecniche della Programmazione, lez. 3

- Approfondimenti PRIMA PARTE

ourProgram.c - (DICHIARAZIONE CON INIZIALIZZAZIONE)

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero=168, secondo=640, ris; /* gli interi …*/



  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu` %d e` %d\n",
                                primoNumero, secondo, ris);

return 0;
}
```

Costanti



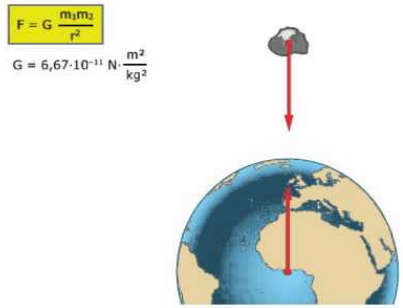
Una costante e` un **IDENTIFICATORE**
che viene associate ad un **VALORE**
prima della compilazione del programma, e che successivamente manterra` quel valore per tutta la durata dell'esecuzione del programma.

conveniente se quel valore appare tante volte nel programma …

e poi magari serve che sia diverso in una versione lievemente diversa del programma.

```
#define PI 3.14159
```

Costanti



Una costante e` un **IDENTIFICATORE**
che viene associate ad un **VALORE**
prima della compilazione del programma, e che successivamente manterra` quel valore per tutta la durata dell'esecuzione del programma.

Ogni volta che, nel programma, viene usato l'identificatore di quella costante, e` il valore corrispondente che viene usato.

L'uso di una costante e` conveniente quando un certo valore deve essere scritto esplicitamente in un programma numerose volte, e magari da questo programma si potrebbe ottenere un altro programma solo variando il valore della costante.

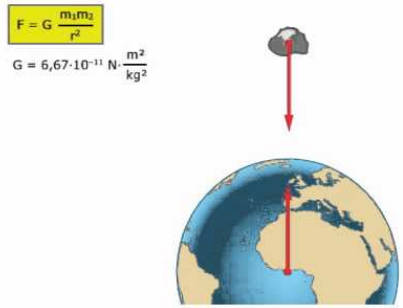
```
#define PI 3.14159
```

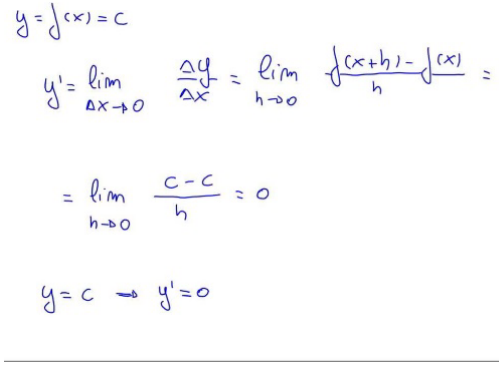
Esercizio …

```
/* programma che chiede e legge da input i
raggi di tre circonferenze e produce per
ciascuna circonferenza la relativa area */
```

Vedi il programma **cerchi.c** … nella nostra geometria PI ha un certo valore (π) e lo usiamo per calcolare l'area di tre circonferenze, usando PI ripetutatmente. Il medesimo programma potrebbe essere usato in una geometria diversa, in cui PI e' diverso, semplicemente cambiando in una sola riga del programma il suo valore… invece che cambiarlo in tutte le istruzioni in cui e' usato.

Costanti e MACRO





Una costante e' un
IDENTIFICATORE
che viene associate ad un **VALORE**
prima della compilazione del programma, e che successivamente manterra` quel valore per tutta la durata dell'esecuzione del programma.

```
#define PI 3.14159
```

- Le costanti vanno definite all'inizio del file con il programma, tipicamente prima della main()
- Gli identificatori sono per convenzione scritti con TUTTE MAIUSCOLE
- #define e` un esempio di direttiva per il compilatore (o MACROISTRUZIONE, o direttamente MACRO)

Le direttive, come #define e #include, vengono gestite dal "precompilatore"; la pre-compilazione e` una fase precedente alla compilazione, in cui in sostanza si prepara il file.c per essere compilato.

e questo e` purtroppo tutto quello che possiamo dire a riguardo qui ☹
approfondimenti sul manuale di C sono possibili ;)

STAMPA

LA FUNZIONE **printf** PERMETTE DI STAMPARE UNA **SEQUENZA DI CARATTERI**, OVVERO UNA **STRINGA**, CHE DEVE ESSERE SPECIFICATA FRA **DOPPI APICI** " ... "

```
printf("STATE IMPARANDO UN SACCO DE ROBBA!");
```

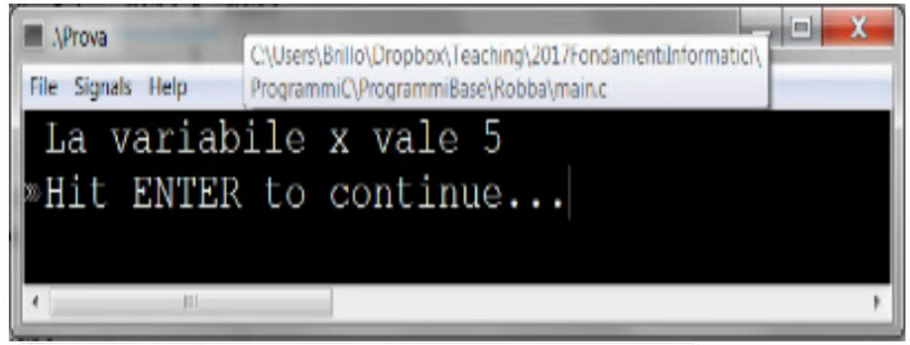
La funzione printf() permette anche di stampare valori, all'interno della stringa di output.
Da adesso la stringa si chiama **stringa di formato**
Per riuscirci, la stringa deve contenere l'indicazione di

1) dove la stampa di un valore deve avvenire lungo la stringa di caratteri;
2) qual è il **formato di conversione** da usare per stampare quel valore (in pratica di che tipo ci aspettiamo che sia il valore)
3) qual è il valore ...

La stampa avviene nel punto in cui appare il formato di conversione.
Il formato di conversione inizia con %
Il valore da stampare appare dopo la stringa di formato, separato da una virgola.
Se ci sono più valori da stampare, devono esserci anche più formati di conversione e ogni valore corrisponde ad uno di essi, in ordine di apparizione.

ESEMPI DI STAMPE

```c
int main() {
    int x = 5;
    printf("La variabile x vale %d", x);
```

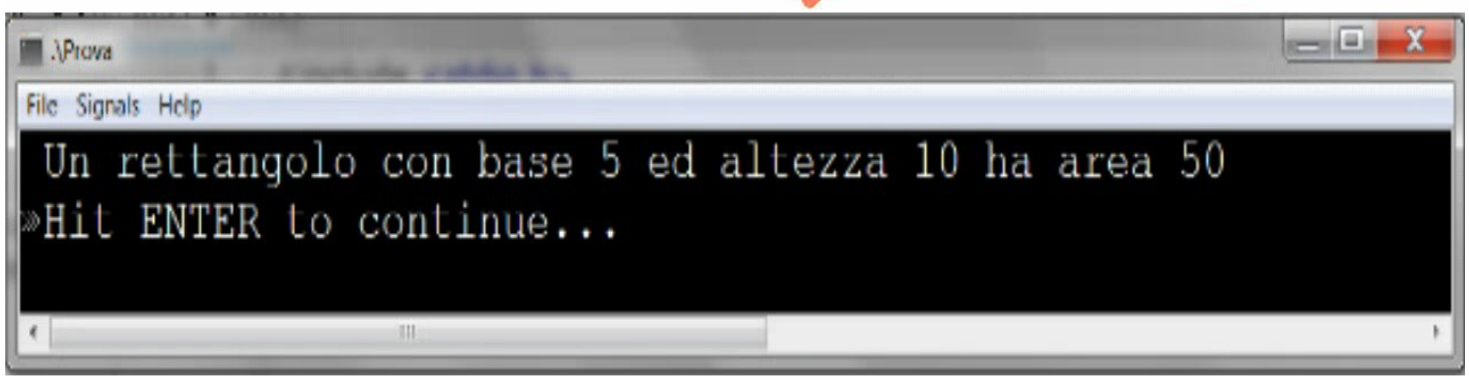


%d È IL FORMATO DI STAMPA PER VALORI DI TIPO int

LA STRINGA E CIASCUN VALORE SONO SEPARATI DA VIRGOLE. PER OTTENERE IL VALORE MEMORIZZATO NELLA VARIABILE X SI SCRIVE SEMPLICEMENTE X (ACCESSO)

```c
int main() {
    int x = 5;
    int y = 10;
    printf("Un rettangolo con base %d ed altezza %d ha area %d", x, y, x*y);
```

I VALORI NON DEVONO ESSERE NECESSARIAMENTE VARIABILI, MA ESPRESSIONI DEL TIPO DEL FORMATO



Caratteri speciali

Alcuni caratteri, espressi con «sequenze di escape», realizzate usando backslash ‘\’, hanno significati speciali

\n	**andata a capo**
\t	**tabulazione (un po’ di caratteri bianchi …)**
****	**stampa **
\’	**stampa ‘**
\’’	**stampa doppio apice**
\b	**back one character**

Assegnazione

E' l'operazione che provoca la MEMORIZZAZIONE di un VALORE nella VARIABILE.

Sintassi

Vai ora alla parte Esercizi per Casa

Tecniche della Programmazione, lez. 3

- Esercizi per casa

esercizi

Nella directory dei complementi didattici per la lezione 03, ci sono risorse per usare il DevC++.

C'e' un "Primer" per iniziare ad usare il DEVC++.
Seguitelo. Quando lo avrete terminato sarete un po' piu' pratici della cosa.
E potrete scrivere altri programmi.

Ricordatevi di salvare sempre i vostri file (se sono programmi in C) con estensione .c (non .cpp).

Dopo il primer potete dare un'occhiata all'ulteriore file pdf disponibile nella directory dei complementi didattici per la lezione 03

Poi lasciate questa directory, per ora e fate gli esercizi suggeriti nelle prossime slide (le cui "soluzioni" sono sempre nella directory dei complementi didattici per questa lezione)

esercizi sui rettangoli e sui cerchi

scrivere rettangolo.c, senza consultare il file rettangolo.c nella directory dei complementi didattici, e nemmeno la slide in cui lo abbiamo discusso.

La specifica del problema è la seguente:

```
/* questo programma, ricevendo in INPUT i lati significativi
   di un rettangolo, ne calcola e stampa in OUTPUT l'area
   */
```

Provare il programma eseguendolo per almeno 5 rettangoli.
In rettangolo.c usare un'unica operazione di INPUT, per leggere i lati.

scrivere rettangolo2.c, identico al precedente, a parte che esegue due distinte operazioni di input, una per il primo lato e una per il secondo.

scrivere un

```
/* programma che chiede e legge da input i raggi di tre circonferenze
   e produce per ciascuna circonferenza la relativa area */
```

Usare una costante PI per il π

POI confrontare la propria soluzione con quella nella directory dei complementi didattici. E correggere quest'ultima che stampa un po' male le cose …

Ripetizione
Quale algoritmo rappresenta ourProgram.c ??

```
il programma ourProgram.c è nella slide successiva
… non c'è bisogno di risalire indietro …

scrivere l'algoritmo completo che corrisponde al programma.

Poi, sì, si può risalire alla slide che abbiamo discusso a lezione e
confrontare quel che si è scritto
```

ourProgram.c

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int primoNumero, secondo;    /* i due interi */
  int ris;                     /* il risultato */

  primoNumero =168;
  secondo = 640;

  ris = primoNumero + secondo;      /* calcolo */

  printf ("il risultato di %d piu' %d e' %d\n",
                               primoNumero, secondo, ris);

return 0;
}
```

ourProgram.c
(riscriverlo, usando la inizializzazione in definizione)

```
/* programma che esegue la somma dei valori contenuti in due variabili intere,
assegnando il risultato ad una terza variabile, che poi viene stampata */
#include <stdio.h>

int main () {
  int
```

```
completare …

facendo uso dell'inizializzazione
in definizione

l'algoritmo è sempre il medesimo!
Cambia solo la tecnica con cui lo
programmiamo …
```

```
return 0;
}
```

Tecniche della Programmazione, lez. 3

- Approfondimenti SECONDA PARTE

Algoritmo (again …)

Abbiamo detto che e` una sequenza di PASSI, operazioni, istruzioni …
per risolvere un problema per il quale abbiamo una formalizzazione matematica

Più schematicamente ci sono delle componenti da considerare

- **INFORMAZIONI** relative al problema: rappresentate come **DATI** (nel calcolatore, nell'algoritmo, nel programma)
- **INPUT** i dati, che distinguono un'**istanza** da un'altra del problema, e che sono da usare nel calcolare la soluzione
- **Procedura Computazionale** passi di calcolo …
- **OUTPUT** …

Progettazione Algoritmo: attivita` creativa …
Esecuzione Algoritmo: attivita` meccanica

Algoritmo (again …)

Abbiamo detto che e` una sequenza di PASSI, operazioni, istruzioni …
per risolvere un problema per il quale abbiamo una formalizzazione matematica
Più schematicamente ci sono delle componenti da considerare

- **INFORMAZIONI** relative al problema : rappresentate come **DATI** (nel calcolatore, nell'algoritmo, nel programma)

- **INPUT** i dati, che distinguono un'**istanza** da un'altra del problema, e che sono da usare nel calcolare la soluzione

- **Procedura Computazionale** questa e` la sequenza di passi …
  1) operazioni su dati di input
  2) operazioni su dati intermedi (ottenuti con calcoli su dati di input e altri dati intermedi)
  3) Produzione di dati di OUTPUT

- **OUTPUT**

Algoritmo (again …)

Abbiamo detto che e` una sequenza di PASSI, operazioni, istruzioni …
per risolvere un problema per il quale abbiamo una formalizzazione matematica
Più schematicamente ci sono delle componenti da considerare

- **INFORMAZIONI** relative al problema : rappresentate come **DATI** (nel calcolatore, nell'algoritmo, nel programma)

- **INPUT** i dati, che distinguono un'**istanza** da un'altra del problema, e che sono da usare nel calcolare la soluzione

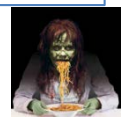
- **Procedura Computazionale** questa e` la sequenza di passi …
  1) operazioni su dati di input
  2) operazioni su dati intermedi (ottenuti con calcoli su dati di input e altri dati intermedi)
  3) Produzione di dati di OUTPUT

- **OUTPUT** i dati emessi a valle della procedura computazionale, interpretabili dall'utente come informazioni sulla soluzione dell'istanza del problema

GO TO and Structured Programming

"go to"

vintage



Ecco due programmi per il problema

Calcolare la somma di numeri interi forniti in input dall'utente.
L'immissione di 0 termina l'input.
I numeri negativi inseriti in input vengono ignorati.

Nel primo programma usiamo goto.
Nel secondo programmazione strutturata, con le istruzioni di controllo.

Dopo aver visto e compreso (anche fatto girare) i programmi,

- confrontare il codice dei due programmi, cercando di giudicare quale e` piu` leggibile.
- scrivere i relativi algoritmi e confrontarli.

Con GO TO

vintage

non la usiamo

Calcolare la somma di numeri interi forniti in input dall'utente.
L'immissione di 0 termina l'input.
I numeri negativi inseriti in input vengono ignorati.

Nel primo programma usiamo goto.

```
#include <stdio.h>

int main() {
    int somma = 0, num;

inizio:
    printf("Inserisci un numero (0 per terminare): ");
    scanf("%d", &num);
    if (num == 0) goto fine;        /* Termina l'input se 0 */
    if (num < 0) goto ignora;       /* Ignora i numeri negativi */
    somma += num;                   /* Somma i numeri positivi */
    goto inizio;

ignora:
    printf("Numero negativo ignorato.\n");
    goto inizio;

fine:
    printf("Somma finale: %d\n", somma);
    return 0;
}
```

Con Istruzioni di controllo della programmazione strutturata

Calcolare la somma di numeri interi forniti in input dall'utente.
L'immissione di 0 termina l'input.
I numeri negativi inseriti in input vengono ignorati.

Nel secondo programma, usiamo programmazione strutturata, con le istruzioni di controllo.

```
#include <stdio.h>

int main() {
    int somma = 0, num;

    while (1) {
        printf("Inserisci un numero (0 per terminare): ");
        scanf("%d", &num);

        if (num == 0) break;          /* Termina l'input se 0 */
        if (num < 0) {                /* Ignora i numeri negativi */
            printf("Numero negativo ignorato.\n");
            continue;
        }

        somma += num;                 // Somma i numeri positivi
    }
    printf("Somma finale: %d\n", somma);
    return 0;
}
```