Programmazione in ANSI C

Standard del 1989 ...

Con successive aggiunte

Le fasi della programmazione

- Ad un primo livello di astrazione l'attività della programmazione puo essere suddivisa in quattro (macro) fasi principali.

1. Definizione del problema (specifica)
2. Individuazione di un procedimento risolutivo (algoritmo)
3. Codifica dell'algoritmo in un linguaggio di programmazione (codifica)
4. Esecuzione e messa a punto (esecuzione e testing)

Il linguaggio C

- I linguaggi direttemente eseguibili dalla macchina sono sequenze binarie difficili da comprendere e manipolare per gli umani (linguaggio macchina)
  - (se riusciamo) vedremo un piccolo esempio
- Tutti i programmi vengono scritti in linguaggi ad alto livello (come il C) che sono leggibili, compatti e modificabili
- Ci sono dei concetti base per la programmazione che si ritrovano in ogni linguaggio ad alto livello, noi li descriveremo in generale esemplificandoli nel linguaggio C

Il linguaggio C

- Prima di essere eseguiti i linguaggi ad alto livello devono essere tradotti in linguaggio macchina (compilazione o interpretazione)
- La traduzione può essere effettuata dal computer utilizzando opportuni programmi (compilatori, interpreti)
  - Più avanti vedremo un po' in dettaglio questo processo relativamente al linguaggio C

Codifica in C

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

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

Lettura/scrittura: `stdio.h`

- Per le interazioni con l'esterno utilizzeremo le funzioni della libreria **`stdio.h`**
- Moltissime funzioni, per ora:
  - **`scanf()`** per leggere dati da tastiera
  - **`printf()`** per scrivere i dati sullo schermo
  - Ne introdurremo le caratteristiche attraverso un alcuni di semplici esempi e nelle prossime lezioni aggiungeremo informazioni e dettagli
- Panoramica più ampia della libreria più avanti

Un primo programma C

Ogni programmatore che si rispetti prova a risolvere il problema di stampare la frase «Ciao Mondo!» sullo schermo.

- Quindi ovviamente ci proviamo anche noi
- In questo caso l'algoritmo è del tutto banale!
- Si tratta di rappresentare in qualche modo la frase e usare le funzioni di libreria **`stdio.h`** per la stampa sullo schermo

Un primo programma C

```
#include <stdio.h>

int main(void)
/* Stampa un messaggio sullo schermo. */
{ printf("Ciao mondo!\n");
  return 0;
}
```

Questo programma stampa sullo schermo una riga di testo:

```
Ciao mondo!
```

Vediamo in dettaglio ogni riga del programma.

Un primo programma C

```
#include <stdio.h>

int main(void)
/* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

E' una direttiva al *preprocessore* un programma di manipolazione testuale che elabora il sorgente prima del compilatore C.
In questo caso serve a ricopiare tutte le informazioni che permettono al compilatore di verificare l'uso corretto delle funzioni della libreria stdio.h (come la printf()...)

Un primo programma C

```
#include <stdio.h>

int main(void)
/* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

E' il prototipo della funzione principale del programma, da cui iniziera' l'esecuzione (si deve chiamare obbligatoriamente **main**) in questo caso non prende nessun argomento (il **void** fra parentesi) e restituisce un valore intero (il **int** prima di **main**) .

In C, la funzione main deve sempre restituire un valore, che indica se è andato tutto bene o no.

Un primo programma C

```
#include <stdio.h>

int main(void)
/* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

Le parentesi delimitano il *blocco* che costituisce il *corpo* della funzione main() cioè l'insieme di *istruzioni* che devono essere eseguiti dalla funzione e le *variabili* utilizzate.

Un primo programma C

```
#include <stdio.h>

int main(void)
/* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

In questo caso abbiamo solo due istruzioni nel blocco, la prima è una chiamata alla funzione di libreria printf() che ha come effetto di visualizzare (stampare) sullo schermo una sequenza di caratteri tra apici.

Un primo programma C

```
#include <stdio.h>

int main(void)
/* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

Una «sequenza di caratteri tra apici» viene comunemenda detta stringa : in questo caso abbiamo una stringa costante, cioè che non muta il suo valore per tutta l'esecuzione del programma.
Il carattere speciale \n corrisponde al ritorno carrello ed ha l'effetto di un'andata a capo.

Un primo programma C

```
#include <stdio.h>

int main(void)
/* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

Più precisamente questa è una istruzione semplice e come ogni istruzione semplice deve terminare con un punto e virgola (;)
I oltre alle istruzioni semplici, esistono anche istruzioni composte (che non richiedono il « ; ») le vedremo più avanti

Un primo programma C

```
#include <stdio.h>

int main(void)
/* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

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

Un primo programma C

```
#include <stdio.h>

int main(void)
/* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

Questo è un commento (racchiuso fra `/* ... */`)
Vedremo durante il corso quali sono le strategie migliori per commentare un programma in modo da aggiungere informazioni che facilitino la comprensione da altri ... (O da chi lo ha scritto dopo qualche tempo)

Un primo programma C

```
#include <stdio.h>

int main(void)
/* Stampa un messaggio sullo schermo. */
{
  printf("Ciao mondo!\n");
  return 0;
}
```

Alcune note:

- In C si usa l'indentazione per rendere i programmi più comprensibili (ma non è essenziale per la semantica, come in Python)
- Il C è case sensitive, printf() è diversa da PRINTF() o PrInTf()

Un primo programma C

Come è possibile mandare in esecuzione il programma appena visto ed ottenere la stampa della stringa sullo schermo ? Dipende dall'ambiente che utilizziamo, nel corso ci riferiremo all'ambiente GNU tipico utilizzando una shell bash. Dovremo

- Compilare : per trasformare quanto abbiamo scritto nel file eseguibile (binario comprensibile alla macchina)
- Eseguire : richiedere alla macchina di caricare il programma in RAM ed istruire il processore adavviare l'esecuzione dalla prima istruzione del main

Lo vedremo in laboratorio .....

Esempio 2: Calcolo dell'area

Cerchiamo adesso di affrontare un problema un po' impegnativo dal punto di vista algoritmico:

- Vogliamo calcolare l'area di un rettangolo
- Leggendo da tastiera i valori di base ed altezza
- E stampando sullo schermo il risultato
- Anche in questo caso utilizzeremo le funzioni della libreria standard per leggere da tastiera gli input e scrivere sullo schermo

Calcolo dell'area : algoritmo

L'algoritmo è molto semplice

- Leggiamo da input il valore della base
- Leggiamo da input il valore dell'altezza
- Calcoliamo l'area (base per altezza)
- Stampiamo il valore dell'area

E' evidente che abbiamo bisogno di qualcosa di nuovo. Dei contenitori per raccogliere i valori letti e per memorizzare il risultato intermedio in attesa di elaborarlo e stamparlo. Ci servono le variabili

Variabili

- Una variabile è un *nome simbolico* collegato ad un *valore*
  - parleremo anche di un contenitore in cui può essere inserito/contenuto/memorizzato un valore
- Molti linguaggi di programmazione (fra cui il C) forniscono la possibilità di definire variabili per contenere i valori necessari al calcolo
- Nel nostro caso ....

Area: algoritmo con variabili

Supponiamo di avere tre variabili di nome simbolico *b*, *h*, *A*

L'algoritmo è molto semplice

- Leggiamo da input il valore della base e inseriamolo in *b*
- Leggiamo da input il valore dell'altezza e inseriamolo in *h*
- Calcoliamo l'area ed assegnamola ad $A = b \times h$
- Stampiamo *A*

Il calcolo dell'area : codifica

- In C è possibile definire variabili per contenere valori di vario tipo, che astraggono la memoria necessaria per rappresentare questi valori
- Ogni variabile deve essere definita esplicitamente all'inizio del blocco che la utilizza, per mezzo di una dichiarazione che specifica il tipo dei valori che deve contenere
- Il C ammette un insieme finito di *tipi predefiniti* per le variabili
  - Nel programma del calcolo dell'area usiamo il **`double`**, uno dei tipi per rappresentare i reali

Calcolo dell'area ... In C

```
#include <stdio.h>

int main(void) {
  double h,b,A;
  printf("Inserisci la base:");
  scanf("%lf",&b);
  printf("Inserisci l'altezza:");
  scanf("%lf",&h);

  A = h * b;

  printf("L'area e': %f\n",A);
  return 0;
}
```

Calcolo dell'area: output

Questo programma stampa sullo schermo una riga di testo:

```
Inserisci la base:
```

Se digitiamo 50.1 e ↓ (invio)

```
Inserisci la base: 50.1
Inserisci l'altezza:
```

Se digitiamo 12.7 e ↓ (invio)

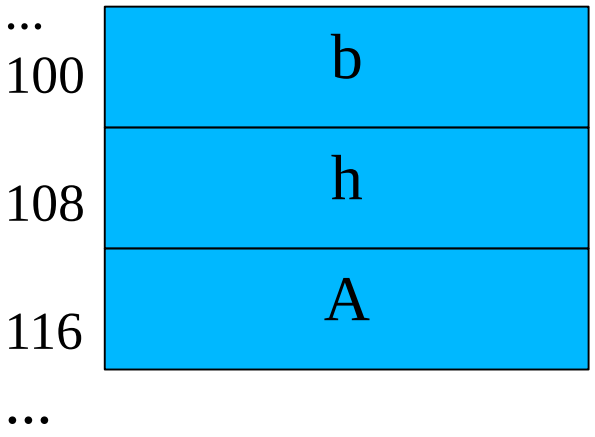
```
Inserisci la base: 50.1
Inserisci l'altezza: 12.7
L'area è: 636.27
```

Calcolo dell'area ... in C

```
#include <stdio.h>

int main(void) {
  double h,b,A;
  printf("Inserisci la base:");
  scanf("%lf",&b);
  printf("Inserisci l'altezza:");
  scanf("%lf",&h);

  A = h * b;

  printf("L'area e': %f\n",A);
  return 0;
}
```

Dichiarazione di 3 variabili

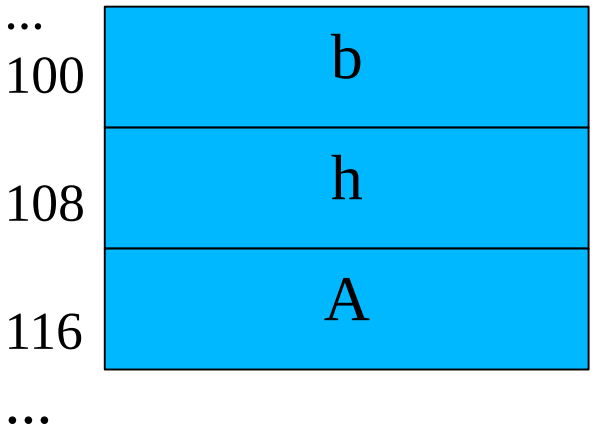
Effetto delle dichiarazioni



Per capire il significato del programma che avete appena visto (la sua *semantica*) è necessario spiegare come vede la memoria un programma C

- Una sequenza di byte, con indirizzi interi positivi
- Ogni variabile corrisponde a un gruppo di byte contigui cui è associato un indirizzo

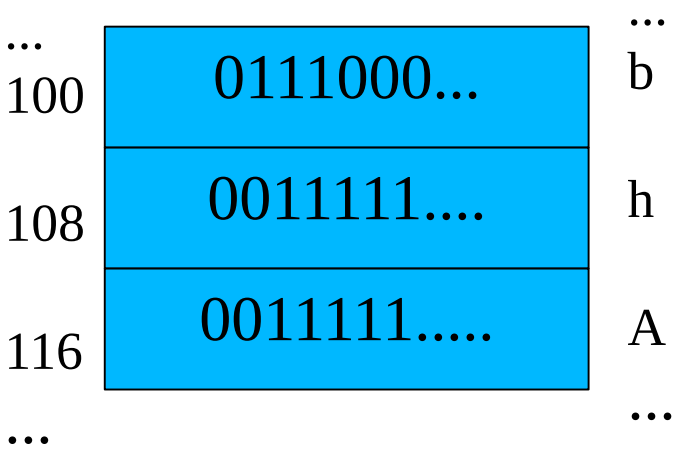
Effetto delle dichiarazioni



Nel nostro caso:

- Per memorizzare un double servono 8 byte
- Per ogni variabile, **`b, h, A`** si riservano 8 byte consecutivi
- Ogni variabile corrisponde ad un indirizzo di memoria quello più basso fra gli indirizzi relativi a quella variabile
- A **`b`** corrisponde l'indirizzo 100

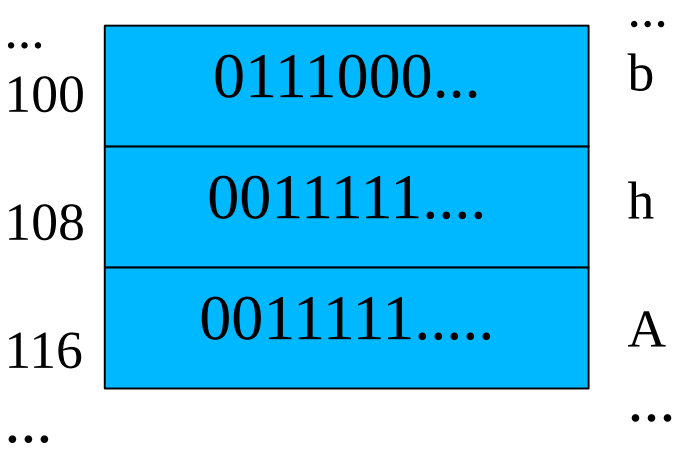
Effetto delle dichiarazioni



La dichiarazione serve solo a riservare lo spazio necessario:

- La memoria riservata contiene valori casuali
- Deve essere inizializzata con valori significativi prima di poterla utilizzare
  - Altrimenti i risultati del codice sono casuali
- Nel nostro caso lo facciamo con le due scanf() e con l'istruzione che effettua il calcolo dell'area...

Effetto delle dichiarazioni



É possibile denotare l'indirizzo di una variabile in C

- Con l'operatore &
- L'indirizzo si chiama puntatore alla variabile
- Ad esempio &b (di valore 100) è il puntatore alla variabile b...
- Ne parleremo più avanti
- Alcune funzioni di libreria (come scanf()) hanno bisogno di avere il puntatore di una o più variabili per andarci a scrivere il valore letto

Calcolo dell'area ... in C

```
#include <stdio.h>

int main(void) {
  double h,b,A;
  printf("Inserisci la base:");
  scanf("%lf",&b);
  printf("Inserisci l'altezza:");
  scanf("%lf",&h);

  A = h * b;

  printf("L'area e': %f\n",A);
  return 0;
}
```

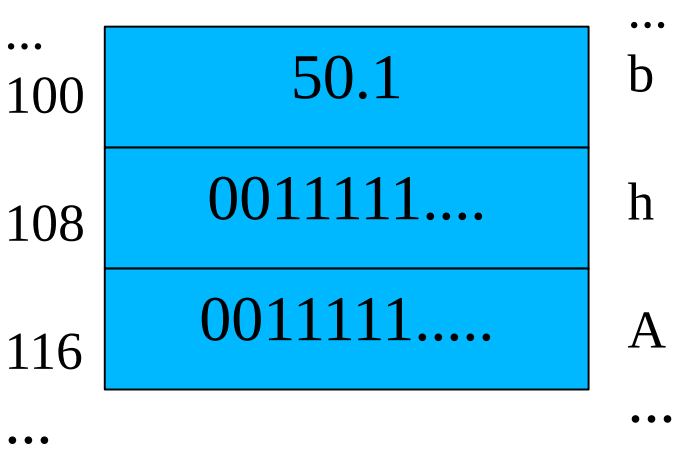
Stampa sullo standard output

Calcolo dell'area ... in C

```
#include <stdio.h>

int main(void) {
  double h,b,A;
  printf("Inserisci la base:");
  scanf("%lf",&b);
  printf("Inserisci l'altezza:");
  scanf("%lf",&h);

  A = h * b;

  printf("L'area e': %f\n",A);
  return 0;
}
```

Lettura (di un double) dallo standard input (viene scritto nella variabile b)

Effetto delle dichiarazioni



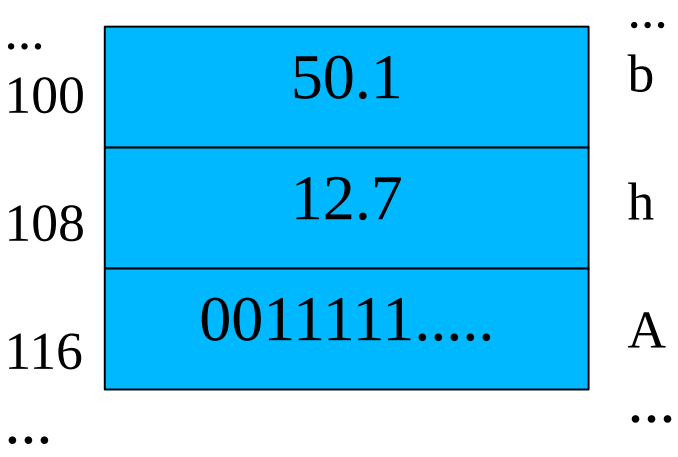
Stato della memoria dopo la prima scanf()

Calcolo dell'area ... in C

```
#include <stdio.h>

int main(void) {
   double h,b,A;
   printf("Inserisci la base:");
   scanf("%lf",&b);
   printf("Inserisci l'altezza:");
   scanf("%lf",&h);

   A = h * b;

   printf("L'area e': %f\n",A);
   return 0;
}
```

Lettura (di un double) dallo standard input (viene scritto nella variabile h)

Effetto delle dichiarazioni



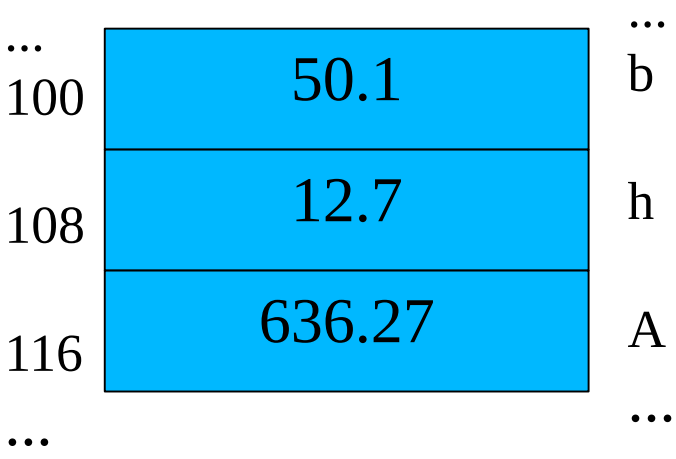
Stato della memoria dopo la seconda scanf()

Calcolo dell'area ... in C

```
#include <stdio.h>

int main(void) {
  double h,b,A;
  printf("Inserisci la base:");
  scanf("%lf",&b);
  printf("Inserisci l'altezza:");
  scanf("%lf",&h);

  A = h * b;

  printf("L'area e': %f\n",A);
  return 0;
}
```

Viene moltiplicato il valore delle variabili h e b ed assegnato alla variabile A

Effetto delle dichiarazioni



Stato della memoria dopo l’istruzione di assegnamento

Calcolo dell'area ... in C

```
#include <stdio.h>

int main(void) {
   double h,b,A;
   printf("Inserisci la base:");
   scanf("%lf",&b);
   printf("Inserisci l'altezza:");
   scanf("%lf",&h);

   A = h * b;

   printf("L'area e': %f\n",A);
   return 0;
}
```

Stampa del valore (double) su standard output

Calcolo dell'area: output

Questo programma stampa sullo schermo una riga di testo:

```
Inserisci la base:
```

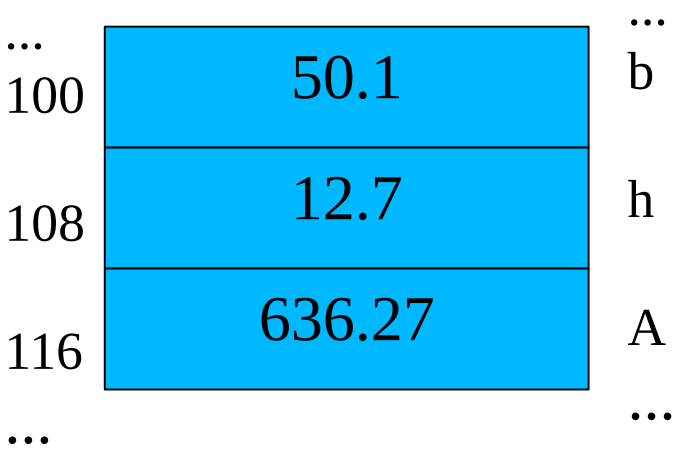
Se digitiamo 50.1 e ↓ (invio)

```
Inserisci la base: 50.1
Inserisci l'altezza:
```

Se digitiamo 12.7 e ↓ (invio)

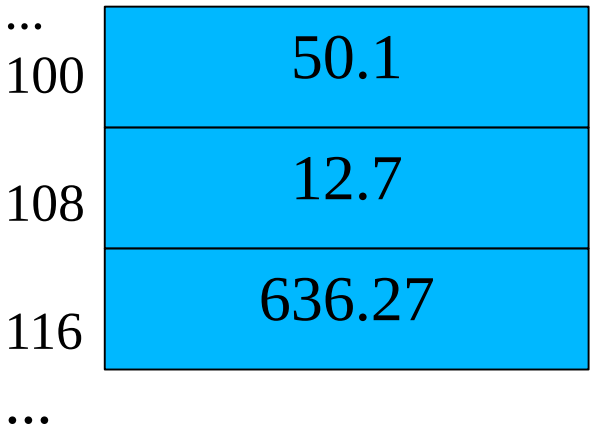
```
Inserisci la base: 50.1
Inserisci l'altezza: 12.7
L'area è: 636.27
```

Ancora sul programma dell'area



L'insieme di tutte le variabili e dei valori a cui sono legate ad ogni istante dell'elaborazione costituisce lo stato del programma

Ancora sul programma dell'area

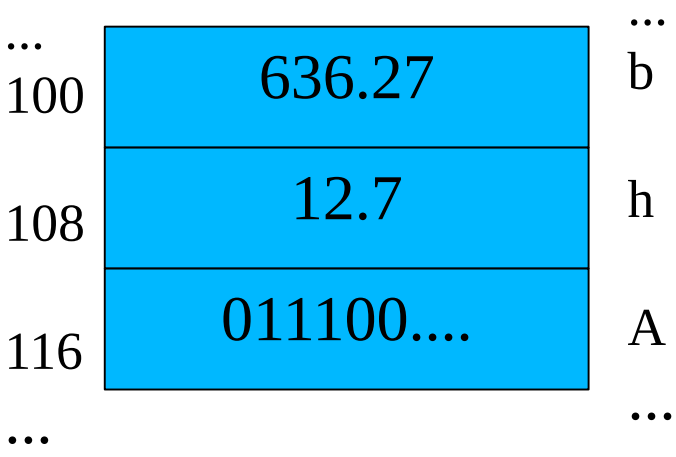


Analizziamo meglio l'istruzione

```
A = h * b;
```

- È un assegnamento
- Valuta l'espressione a destra
  - Estrae i valori delle variabili a destra (h e b)
  - Applica gli operatori ed ottiene il valore risultante (636.27)
- Assegna il valore risultante alla variabile di sinistra (A)

Ancora sul programma dell'area



Avremmo potuto usare la stessa variabile a sinistra e a destra

```
b = h * b;
```

- E stampare **b** invece di A
- Questo perchè la valutazione del valore di **b** a sinistra avviene prima di applicare l'operatore
- Solo dopo la fine della valutazione si modifica il valore di **b** in memoria

Riassumiamo: le variabili

- Servono a rappresentare, nei programmi, le associazioni (modificabili) dello stato
- Una variabile in C è caratterizzata da:
  1. nome: serve a identicarla
     |Es: A,b...
  2. valore: valore associato nello stato corrente
     Es: 4 (puo cambiare durante l'esecuzione)
  3. tipo: specifica l'insieme dei possibili valori
     |Es.: double (reali), int (interi)
  4. indirizzo/puntatore: della cella di memoria a partire dal quale è memorizzato il valore.

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

Riassumiamo: assegnamento

- Serve a modificare il valore di una variabile (e quindi lo stato del programma)
- L’uguale (=) è l’operatore di assegnamento
- a destra di "=" possono comparire espressioni
- il valore assegnato è quello dell'espressione calcolata nello stato corrente. Una variabile all'interno di una espressione sta per il valore ad essa associato in quel momento

Assegnamento

- Esempi di istruzioni di assegnamento:

```
a = a + 4;
b = b + 1;
c = (a * b) / 3;
d = h % n;
```

- Nelle espressioni a destra c'è una implicita precedenza fra gli operatori (come avviene normalmente nelle espressioni matematiche), quindi è importante utilizzare le parentesi per guidare l'ordine di valutazione

Assegnamento

- Quando l'espressione coinvolge una singola variabile, come ad esempio :

```
a = a * 4;
b = b + 1;
d = d % n;
```

...è possibile utilizzare l'abbreviazione seguente che combina l'operatore utilizzato con il simbolo uguale

```
a *= 4;
b += 1;
d %= n;
```

Assegnamento

- Inoltre quando l'espressione è un incremento o decremento di 1 come in:

```
a = a - 1;
b = b + 1;
```

...è possibile utilizzare le abbreviazioni seguenti

```
a--; --a;
b++; ++b;
```

che sono tutte equivalenti a meno di non utilizzarle dentro un'altra espressione come in

```
a = (++b) - (a--);
```

...cosa assolutamente sconsigliata in ogni caso ....

Funzione printf(): formato

```
printf("Area: %lf\n", A);
```

- il primo argomento è la stringa di formato che puo contenere i *placeholder* (specificatori di formato), introdotti dal carattere `%`
- Uno diverso per ogni tipo (`%lf` indica che deve essere stampato in formato double)
- Il valore stampato è quello della variabile specificata come secondo argomento

Funzione printf(): formato

- è una funzione variadica (numero variabile di argomenti)
- Si possono specificare più placeholder

Es:

```
printf("%lf %d %x", i1, i2, i3);
```

- Ogni placeholder viene associato alla corrispondente variabile
- In questo caso si stampa un double (**`%lf`**) un intero in formato decimale (**`%d`**) ed un intero in formato esadecimale (**`%x`**)
- Ci sono moltissimi formati, ne parleremo quando introdurremo i tipi del C
- La lista completa nel manuale in linea (**`man 3 printf`**)

Funzione scanf(): formato

```
scanff("%f", &b);
```

- il primo argomento è la stringa di formato (simile a printf()) che puo contenere i *placeholder* (specificatori di formato), introdotti da `%`
- Uno diverso per ogni tipo (`%f` indica che deve letto il valore di una variabile **`double`** o **`float`**)
- Il valore letto è trasformato nella rappresentazione binaria corrispondente e scritto nella variabile di puntatore **`&b`** (specificata come secondo argomento)
- È variadica come la **`printf()`**

Nota di struttura di un programma C

```
/*DIRETTIVE al PREPROCESSORE */
#include <stdio.h>
/*FUNZIONI (in questo caso solo main */
int main(void) {/* INIZIO BLOCCO */
  /* PARTE DICHIARATIVA */
  double h,b,A;
  /* PARTE ESECUTIVA */
  printf("Inserisci la base:");
  scanf("%lf",&b);
  printf("Inserisci l'altezza:");
  scanf("%lf",&h);
  A = h * b;
  printf("L'area e': %f\n",A);
  return 0;
}
```

Nota di struttura....

- Le variabili che servono vanno tutte dichiarate nella parte dichiarativa
  - ANSI C, in realtà gcc (c99) le accetta anche dopo
- Questo permette di rilevare più errori in fase di compilazione
  - Variabili usate e mai dichiarate indicano un’anomalia

Ancora su dichiarazione di variabili

- È possibile inizializzare le variabili al momento della dichiarazione

  Es: **`double a=0, b, A=1;`**

- È possibile dichiarare delle variabili «costanti», cioè che non possono essere modificate in fase di esecuzione

  Es: **`const double pigreco=3.14;`**

A che servono le costanti

- A paramentrizzare il codice rispetto ad un valore
  - Se lo voglio cambiare devo agire in un solo punto
  - È più leggibile
- In C ci sono due modi standard di paramentrizzare il codice con valori constanti
  - Le dichiarazioni di variabili **`const`**
  - Le macro (**`#define`**)
  - Ne discuteremo e li confronteremo più avanti

Facciamo un passo avanti

- Per scrivere programmi C più significativi bisogna introdurre
  - I costrutti di controllo, che permettono scegliere azioni alternative in base a cosa sta succedendo nel programma o di ripetere azioni
  - I tipi di dato, che ci dicono quali rappresentazioni per i dati sono messe a disposizione dal linguaggio C