Il Linguaggio C

Prof. Orazio Mirabella

Approccio Top-down e Bottom-up

- Consiste nel suddividere un problema in sottoproblemi via via più semplici.
- Raggiunto un livello sufficientemente elementare si passa alla implementazione dei moduli.
- L'approccio complementare è il "Bottom-up in cui prima si risolvono le singole parti di un problema e poi si compone la soluzione.
- L'approccio Top-down e Bottom- up possono coesistere

Programmazione modulare

- Il progettista affida ad ogni modulo (blocco di istruzioni) un determinato compito.
- Il modulo può essere riutilizzato ogni volta che ce ne sia bisogno.
- Riusabilità
- I moduli debbono interagire fra loro: è necessaria una rete di interazioni
- Modulo = Sottoprogramma.

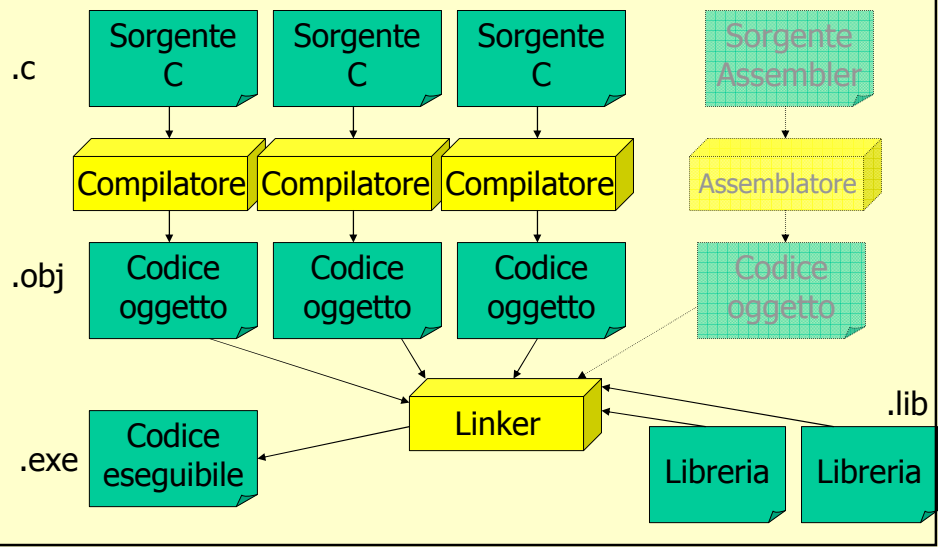
Perché usare il linguaggio C?

- E' un linguaggio ad **alto livello**
- Consente di effettuare **operazioni a basso livello**
- Nei sistemi UNIX è sempre presente un **compilatore C**
- Gli eseguibili sono **veloci**
- E' abbastanza **portatile**

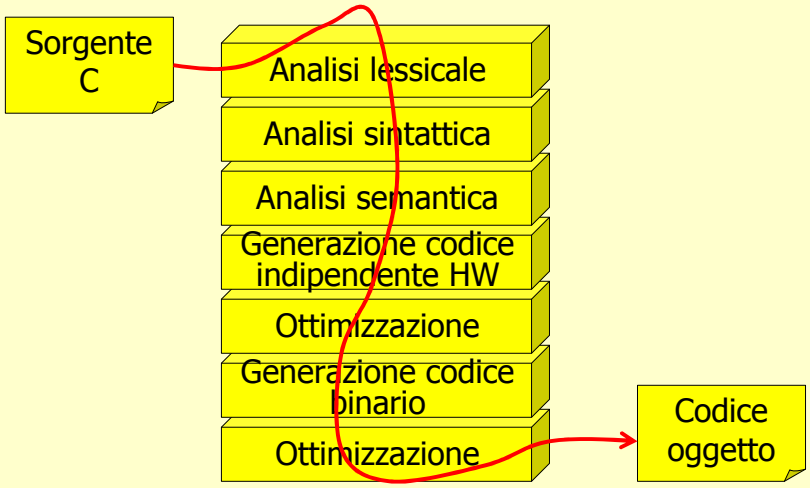
Programmazione modulare

- Non è possibile creare programmi di elevata complessità lavorando con un unico file sorgente
  - Lentezza di ricompilazione
  - Difficile riuso di procedure
  - Impossibile collaborare tra più programmatori
  - Difficile tener traccia delle modifiche
- Occorre realizzare programmi distribuiti su più file sorgenti.

Compilazione e Link



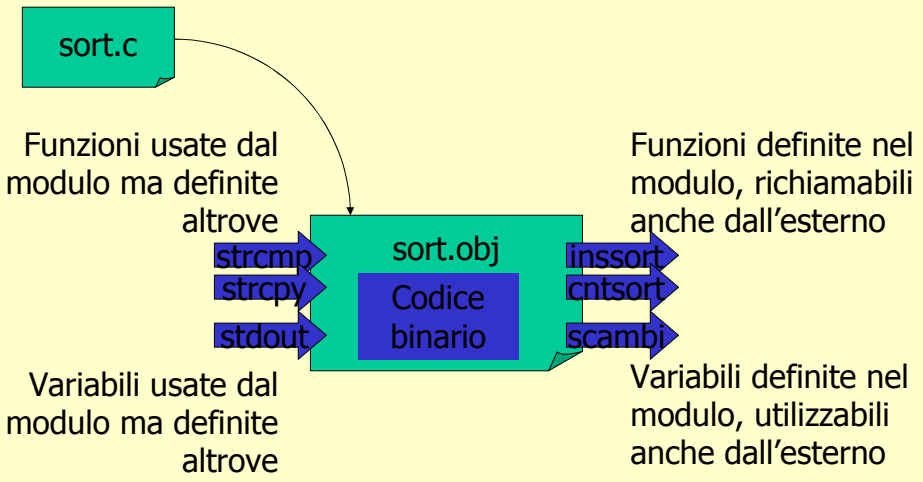
Struttura del compilatore



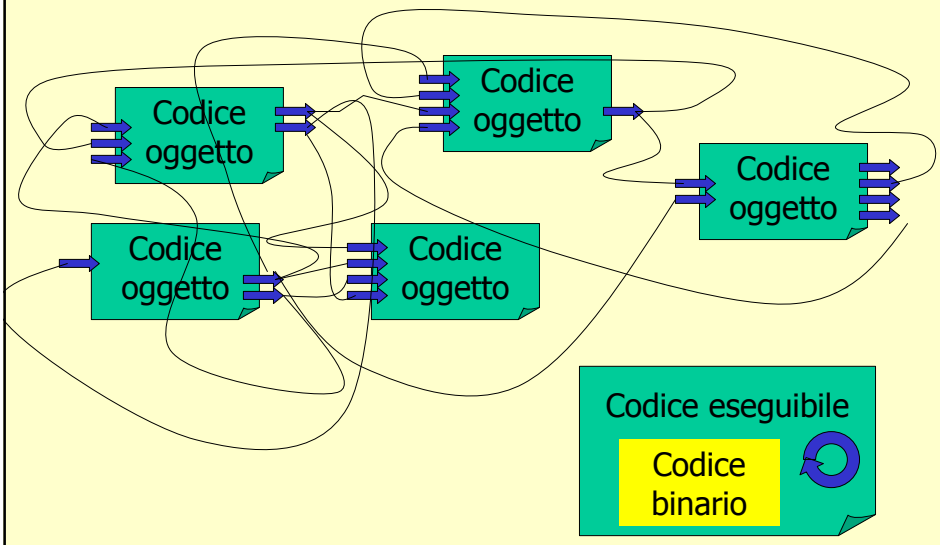
Modulo di Codice oggetto

- Contiene il codice binario corrispondente ad un *modulo* C
- Contiene dei *riferimenti esterni* a funzioni e variabili dichiarate in altri moduli
- Contiene funzioni e variabili utilizzabili da altri moduli
- Uno ed un solo modulo contiene main().

Modulo oggetto



Linker

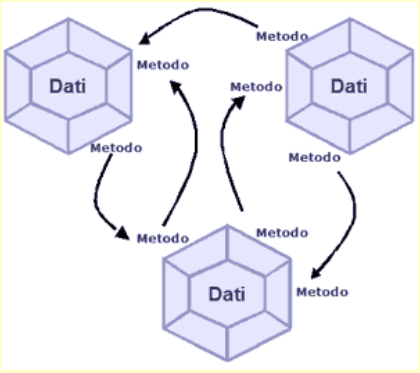


Modularità

- Ciascun modulo in C:
  - ➔Definisce alcune funzioni “private”
  - ➔Definisce alcune funzioni “pubbliche”
  - ➔Definisce alcune variabili “private”
  - ➔Definisce alcune variabili “pubbliche”
  - ➔Chiama alcune funzioni “esterne”
  - ➔Usa alcune variabili “esterne”.

Programmazione orientata agli oggetti

E’ basato sul fatto che esistono una serie di oggetti che interagiscono vicendevolmente, scambiandosi messaggi ma mantenendo ognuno il proprio stato ed i propri dati.



Motivazioni

- Migliorare la qualità del software
- I programmi di grandi dimensioni vengono scomposti in moduli, che chiameremo oggetti
- Ne trae beneficio la fase di manutenzione
- Riutilizzo del codice

Programmazione ad oggetti

- Oggetti
- Classi
- Interazione tra oggetti
- Incapsulamento
- Interfaccia
- Accesso agli attributi

Oggetti

- Per capire cos'è un oggetto prendiamo spunto dalla vita reale: un oggetto è un'automobile, un computer, una casa, e così via
- Un oggetto può essere definito elencando sia le sue caratteristiche sia i suoi comportamenti, cioè le funzioni che mette a disposizione
- Elencando le caratteristiche e i comportamenti, diamo una definizione generica
- Un oggetto però è un'entità particolare, con le sue caratteristiche specifiche.

Oggetti

- Ad un preciso oggetto assegniamo un nome: chiameremo quindi un oggetto di classe automobile: *auto1*

- Quindi l'oggetto *auto1* potrà essere così descritto:
  - Velocità = 80
  - Colore = rosso
  - Numero di porte = 5
  - Marca = bmw

Oggetti

- Un altro oggetto, *auto2,* potrà avere le seguenti caratteristiche:
  - Velocità = 60
  - Colore = nero
  - Numero di porte = 4
  - Marca = fiat
- Quindi gli oggetti sono diversi a seconda del valore assunto dalle loro caratteristiche, ma condividono la stessa struttura

Oggetti

- Nella terminologia OOP le caratteristiche di un oggetto vengono chiamate attributi, i comportamenti vengono chiamati metodi
- I metodi sono le operazioni che un oggetto è in grado di compiere
- Un oggetto è quindi formato da attributi e metodi

Classi

- Possono esistere più oggetti che hanno gli stessi attributi, anche con valori diversi, e che dispongono degli stessi metodi. Si dice che questi oggetti appartengono alla stessa classe
- Una classe specifica gli attributi, senza indicarne il valore, e i metodi che devono avere gli oggetti che appartengono alla classe
- La classe quindi crea più oggetti tutti con gli stessi attributi e gli stessi metodi. Gli oggetti creati a partire da una classe vengono chiamati istanze della classe
- Due istanze della stessa classe sono distinguibili solo per il valore dei loro attributi, mentre il loro comportamento (metodo) è lo stesso

Classi

- La struttura di una classe è la seguente:

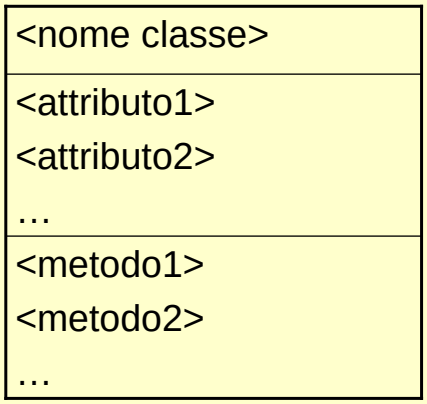


Diagramma delle classi

Classi

- Esempio:

Automobile
Velocità Colore Numero di porte Marca
Avviati Accelera Fermati Gira Cambia marcia Rifornisciti

Programmazione: Prerequisiti

- Concetto di **variabili** e **costanti**
- Concetto di **algoritmo**
- I **diagrammi a blocchi**
- Concetto di **linguaggio** di programmazione
- Concetto di **compilatore**

Il Primo Programma

- Programma che stampa sullo schermo la frase "*Ciao mondo!*"

```
#include <stdio.h>
main()
{
   printf("Ciao mondo!\n");
}
```

Analisi del programma: #include

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

- Questa istruzione permette di includere le informazioni presenti nel file stdio.h
- **Stdio.h**
  - è un file speciale che contiene la descrizione delle funzioni di Input/Output
  - **Deve essere incluso** in tutti i programmi che usano tali funzioni

#include è una direttiva del preprocessore, un comando, che permette di richiamare le librerie standard del C. Senza librerie un programma non avrebbe a disposizione i comandi per eseguire anche le operazioni più semplici

Analisi del programma: `main()`

```
#include <stdio.h>
main()
{
   printf("Ciao mondo!\n");
}
```

Il corpo principale del programma è inserito all'interno della funzione **main()**.

Ogni programma C deve avere uno e un solo **main()**

Le parentesi graffe servono, per delimitare blocchi di istruzioni, o come vengono abitualmente chiamate "statement", che sono eseguite in ordine, da quella più in alto, giù fino all'ultima;
Il punto e virgola, invece, serve per "chiudere" un'istruzione, per far capire al compilatore che dopo quel simbolo inizia una nuova istruzione.

Analisi del Programma: `printf`

```
#include <stdio.h>
main()
{
   printf("Ciao mondo!\n");
}
```

- `Printf` funzione per stampare su video sequenze di caratteri (racchiusi tra " ")
- `\n`
  - ➔ Carattere di new line (ritorno a capo)

Cosa Stampa?

```
#include <stdio.h>
main()
{
   printf("Ciao,\n");
   printf("mondo!\n");
}
```

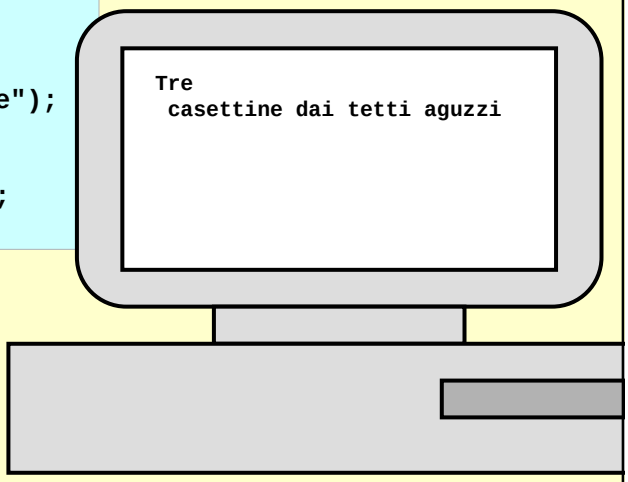

```
#include <stdio.h>
main()
{
   printf("Ciao, ");
   printf("mondo!");
   printf("\n");
}
```

Cosa Stampa?

```
#include <stdio.h>
main()
{
   printf("Tre");
   printf("casettine");
   printf("dai");
   printf("tetti");
   printf("aguzzi");
}
```

Cosa Stampa?

```
#include <stdio.h>
main()
{
   printf("Tre\n");
   printf(" casettine");
   printf(" dai");
   printf(" tetti");
   printf(" aguzzi");
}
```



Commenti

- All'interno di un programma C possono essere inseriti dei commenti, basti sapere, per adesso, che esistono due modi:
- // Tutto quello che sta a destra sulla medesima riga viene considerato commento e viene ignorato (ai fini dell'interpretazione del linguaggio) dal compilatore;
- /* … */ Tutto quello che è compreso tra i due asterischi viene considerato commento; questa forma viene utilizzata per commenti su più righe.

Struttura di un programma C

```
#include <stdio.h>
main()
{
        int a;
        int b;
        int s;

        scanf("%d %d", &a, &b);
        s=a+b;
        printf("%d", s);
}
```

intestazione

dichiarazioni

sezione esecutiva

Variabili

- le variabili sono dei contenitori, identificati da un nome **univoco**, di un qualsiasi valore, sia esso un numero o una stringa.
- Nella variabile, la corrispondenza nome-valore permette di gestire i cambiamenti di valore ad esse associati, fornendo quindi quella dinamicità necessaria ad eseguire operazioni complesse e/o ripetitive con il minimo sforzo.
- Le variabili vengono definite da un **tipo** e da un **nome**.
- Il nome per identificare una variabile (o una funzione o una costante) viene comunemente riferito come **identificatore**.
- Un identificatore è costituito da una o più lettere, cifre o caratteri e deve iniziare con una lettera o il carattere di sottolineatura (underscore “_”); la loro lunghezza massima dipende dal compilatore 255-32-8 caratteri.

Attenzione!

- Il **C** distingue tra lettere maiuscole e lettere minuscole

  ➔Scrivere **`MAIN()`** o **`Main()`** invece che **`main()`** **`è un`** **Errore!**

  ➔La variabile **`Numero`** è diversa dalla variabile **`NUMERO`** e dalla variabile **`numero`**

Un esempio

```
/* Calcolo area rettangolo */
#include <stdio.h>
main()
{
  int base;
  int altezza;
  int area;
  base = 3;
  altezza = 7;
  area = base*altezza;
  printf("%d\n", area);
}
```

Stamperà: 21

- Allo scopo di rendere più chiaro il risultato dell'esempio precedente, si possono visualizzare i valori delle variabili base e altezza:

```
printf("%d ", base);
printf("%d ", altezza);
printf("%d", area);
```

Stamperà: 3 7 21

Proviamo a usare le Costanti

- Evita di scrivere più volte un valore che non cambia
- **Non** ci sono **difformità** tra le varie occorrenze
- Migliora la **leggibilità** dei programmi
- Aiuta il programmatore a **Riutilizzare** i programmi (parametrizzazione)

Esempio: Un programma che manipola matrici quadrate di dimensione 100 può essere facilmente riutilizzato per le matrici di dimensione 200, se tale dimensione è rappresentata da una costante

Usiamo le Costanti

- Le costanti usate da un programma vengono dichiarate nel seguente modo:

```
#define nome valore
```

Esempi:

```
#define IVA 21
#define VAL 0xAF
#define RATA 500
```

Esempio: costanti

```
#include <stdio.h>
#define PI_GRECO 3.14
main()
{
   float raggio, area;
   printf("raggio: ");
   scanf("%f", &raggio);
   area = PI_GRECO * raggio * raggio;
   printf("Area = %f\n", area);
}
```

La definizione di una costante implica implica che il suo valore non può essere modificato.

Ricalcoliamo l'area del rettangolo

```
/* Calcolo area rettangolo, prova utilizzo costanti */
#include <stdio.h>
#define BASE 3
#define ALTEZZA 7
main()
{
  int area;
  area = BASE * ALTEZZA;
  printf("Base: %d\n", BASE);
  printf("Altezza: %d\n", ALTEZZA);
  printf("Area: %d\n", area);
}
```

Tipi di Dati

- Il C supporta diversi tipi di dati.

Tipo	Dimensione
char	1 byte
Short	2 byte
int	4 byte
long	4 byte
float	4 byte
double	8 byte
long double	8 byte

Dichiarazioni

- Tutte le variabili usate in un programma devono essere dichiarate nella zona dichiarativa secondo la sintassi:

***tipo* nome;**

- Esempi:

```
• int eta;
• float statura;
• double AreaRettangolo;
• boolean risposta;
```

Dichiarazioni

- Si può anche usare un'unica dichiarazione per diverse variabili dello stesso tipo:

***tipo* nome1, nome2, nome3;**

- Esempio:

```
int a;
int b;
int s;
```

⟺

```
int a,b,s;
```

Inizializzazione di una variabile

- E' possibile anche in fase di dichiarazione, assegnare alla variabile un valore iniziale:

***tipo* nome = *valore*;**

- Esempio:

```
int contatore = 0;
float area = 4.8;
```

Tipo Intero

- Viene utilizzato per tutte le grandezze che possono essere rappresentate come numeri interi

Esempi:
- età,
- numero di figli
- Punteggio di una partita
- posti auto
- …

Campo di Variabilità

Tipo	Dimensione (byte)	Valore minimo	Valore massimo
`short int`	2	-32768	32767
`int`	4	-2^{31}	$2^{31}-1$
`long int`	4	-2^{31}	$2^{31}-1$

- NOTA: alcuni compilatori usano 2 byte invece di 4 per memorizzare un int

Dettagli sul Tipo Intero

- il qualificatore **unsigned** consente alla variabile di contenere solo numeri positivi

Tipo	Byte	Valore minimo	Valore massimo
`unsigned short int`	2	0	65535
`unsigned int`	4	0	$2^{32}-1$
`unsigned long int`	4	0	$2^{32}-1$

Notazione per valori costanti

- Sequenza di cifre preceduta eventualmente da + o –
- Basi supportate
  - **Decimale**
  - **Ottale** (basta fare precedere la cifra 0 al numero)
  - **Esadecimale** (basta aggiungere il prefisso 0x al numero)

Esempi:

```
627
-325
+29

027 /* ottale */
0x1E5F /* esadecimale */
```

Tipo Reale

- Il tipo reale viene utilizzato per rappresentare prezzi, pesi, misure, per calcoli matematici ecc.

Esempi:
- area di un cerchio,
- prezzo di un articolo
- Media di voti
- …

Campo di Validità

Tipo	Dimensione (byte)	Valore minimo	Valore massimo
`float` Precisione singola	4	$-3.2\times10^{\pm38}$	$+3.2\times10^{\pm38}$
`double` Precisione doppia	8	$-1.7\times10^{\pm308}$	$+1.7\times10^{\pm308}$

Notazioni

- Esistono due modi per scrivere numeri reali

```
<Parte intera>.<Parte decimale>
```

✓Oppure:

```
<Parte intera>.<Parte decimale>E<esponente>
```

Esempi:

- **4.34**
- **-3E3** rappresenta -3×10 elevato a 3 cioè -3000
- **5.7E-2** rappresenta 5.7×10 elevato a -2 cioè 0,057

Operatori

- Somma **+**
- Differenza **–**
- Prodotto ***
- Divisione **/**
- Resto della divisione intera (modulo) **%**
- Incremento **++**
- Decremento **–**
- Assegnazione **=**, **+=**, **-=**, ***=**, **/=**, **%=**

Esempio: divisione e modulo

```
int x = 9;
int y = 6;



x / y                    ------>    vale 1

x % y                    ------>    vale 3
```

- **NOTA**:
- Se `y` vale `0` allora x/y e x%y generano un errore!
- Se `x % y` vale `0` allora `x` è un multiplo di `y`

Esempio: incremento e decremento

```
int x = 9;
int y = 6;

 x++;                    ------>    x vale 10

 y--;                    ------>    y vale 5
```

Assegnazione

- Assegnare un valore ad una variabile:

```
nome = valore;
```

- valore può essere:
  - ➔una costante
  - ➔una variabile
  - ➔espressione

Esempi:

```
x=3;
x=y;
delta=0.001;
somma=a+b;
area = (base * altezza)/ 2;
```

Assegnazione

- E' possibile avere delle espressioni del tipo:

```
conta = conta +1;
```

- nota: la variabile `conta` compare a sia a sx che a dx del segno =
- il nuovo valore di `conta` si ottiene aggiungendo 1 al precedente valore;

Esempi:

```
int c = 8;
int d = 9;
c = c-1;   //adesso c vale 7
d = d*2;    //d vale 18
```

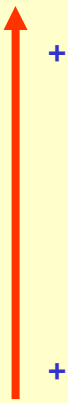
Gli altri operatori di assegnazione

Operatore	Utilizzo	Esempio	Istruzione equivalente
+=	Incremento	x+=y	x=x+y;
-=	Decremento	x-=y	x=x-y;
*=	Moltiplicazione	x*=y	x=x*y;
/=	Divisione	x/=y	x=x/y;
%=	Resto della divisione intera	x%=y	x=x%y;

Precedenze

- Per le operazioni aritmetiche le precedenze sono quelle definite in matematica

Operatori
Parentesi **()**
Operatori unari **- ++ --**
Operatore binario *** / %**
Operatori binari **+ -**
Operatori di assegnazione **= +=, -=, *=, /=, %=**



+ alta

+ bassa

Una funzione base: printf

- La funzione printf permette di visualizzare i dati sul display.
- Per visualizzare un intero;

printf ("%d", nome_variabile);

- dove "d" è detto carattere di conversione o specificatore di formato

Esempio;

```
int x;
int somma=0;
…
printf("%d", x);
printf("%d", somma);
```

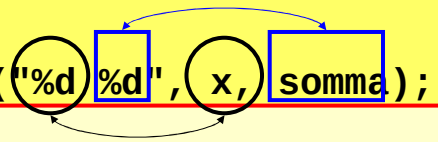
printf

- Se vogliamo stampare più interi con una printf:

printf ("%d %d", nome1, nome2);

Esempio;

```
int x;
int somma=0;
…..
x=7;
printf("%d %d", x, somma);
```



printf

- Per lasciare una linea vuota si deve inserire uno \n nell'istruzione printf all'interno di doppi apici:
  es: printf("AREA RETTANGOLO\n\n");
- Il primo \n fa andare il cursore a linea nuova dopo la visualizzazione di AREA RETTANGOLO, il secondo lo fa scorrere di un ulteriore linea.
- Il ragionamento è valido in generale: se si desidera saltare un'altra riga basta aggiungere un \n e se si vuole lasciare una linea prima della visualizzazione si fa precedere \n ad AREA RETTANGOLO:
  es: printf("\nAREA RETTANGOLO\n\n\n");

printf

- Così come \n effettua un salto a linea nuova, la sequenza \t provoca l'avanzamento del cursore di uno spazio di tabulazione:

Es: printf("Base: %d\tAltezza: %d\tArea: %d", base, altezza, area);

produce come uscita:

Base: 10 Altezza: 13 Area: 130

Sequenze di escape:
\n va a linea nuova
\t salta di una tabulazione
\b ritorna un carattere indietro (*backspace*)

scanf

- La funzione scanf permette di acquisire dati da tastiera:
- Per un intero:

scanf ("%d", &nome_variabile);

- nota: il nome della variabile è preceduto dal simbolo **&**

Esempio;

```
int x;
int somma=0;
scanf("%d", &x);
scanf("%d", &somma);
```

Esempio: scanf e printf

```
#include <stdio.h>
main()
{
   int x;

   printf("Inserisci un numero intero: ");
   scanf("%d", &x);
   printf("Hai inserito %d\n", x);
}
```

Esercizio

- Scrivere un programma che chieda all'utente di inserire un numero rappresentativo di una durata temporale espressa in minuti, e lo converta nel formato <ore> h, <min> m
- Esempio
  - ➔Utente immette 134 minuti → 2 h, 14 m
  - ➔Utente immette 45 minuti → 0 h, 45 m
  - ➔Utente immette 180 minuti → 3 h, 0 m

Soluzione

```
#include <stdio.h>
main()
{
   int tempo, minuti, ore;

   printf("Dammi il tempo in minuti: ");
   scanf("%d", &tempo);
   ore = tempo / 60;
   minuti = tempo % 60;
   printf("%d h, %d m", ore, minuti);
}
```

Funzioni

- Una *funzione* è costituita da un insieme di istruzioni che realizzano un compito: a partire da uno o più valori presi in input, essa restituisce un determinato valore in output.

- Esistono funzioni predefinite o standard, già pronte all'uso, che il linguaggio mette a disposizione del programmatore.

Es: w = abs(j);

assegna a w il valore assoluto di j. All'interno delle parentesi tonde può essere inserito direttamente un valore, come nel caso di w = abs(-186); che assegna a w il valore 186.

Per poter utilizzare tale funzione si deve dichiarare esplicitamente nel programma, prima del main, l'inclusione del riferimento a tale libreria: #include <math.h>

Calcolo della lunghezza di un segmento

```
/* Esempio utilizzo di abs() */
#include <stdio.h>
#include <math.h>
main()
{
  int a, b, segmento, lunghezza;
  printf("\n\nLUNGHEZZA SEGMENTO\n");
  printf("Primo estremo: ");
  scanf("%d", &a);
  printf("Secondo estremo: ");
  scanf("%d", &b);
  segmento = a-b;
  lunghezza = abs(segmento);
  printf("Lunghezza segmento: %d\n", lunghezza);
}
```