



# Una panoramica sul C

Francesco Isgrò




# Cosa vedremo

- Una veloce panoramica del linguaggio C
- Vedremo dei programmi di esempio e ne spiegheremo gli elementi
- Introduurremo brevemente i principali costrutti
- Approfondimenti durante il resto del corso



# Output

- Un programma deve comunicare per essere utile
- Un programma produce sempre un output di qualche tipo
- Il primo esempio che vediamo è un programma che stampa una frase a schermo



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

```
int main(void)
```

```
{
```

```
    printf("from sea  to shining C\n");
```

```
    return 0;
```

```
}
```



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

```
int main(void)
```

```
{
```

```
    printf("from sea  to shining C\n");
```

```
    return 0;
```

```
}
```

- Scriviamo il testo su un file sea.c
- Passi successivi
  - Compilare il programma
  - Mandare in esecuzione il file eseguibile creato



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

```
int main(void)
{
    printf("from sea  to shining C\n");

    return 0;
}
```

- Compilazione
  - gcc sea.c
  - Se non ci sono errori viene creato il file a.out
- Esecuzione
  - ./a.out
- Vediamo in pratica



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

```
int main(void)
{
    printf("from sea to shining C\n");

    return 0;
}
```

- Un preprocessor viene implicitamente eseguito prima della compilazione
- Linee di codice che iniziano con # sono direttive per il preprocessor
- Questa richiede di includere una copia dell'header file in quel punto del codice



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

```
int main(void)
{
    printf("from sea to shining C\n");

    return 0;
}
```

- Un preprocessor viene implicitamente eseguito prima della compilazione
- Linee di codice che iniziano con # sono direttive per il preprocessor
- Questa richiede di includere una copia dell'header file in quel punto del codice
- **N.B.** Boccio chiunque dica che include la libreria!!!!





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

```
int main(void)
```

```
{  
    printf("from sea to shining C\n");
```

```
    return 0;
```

```
}
```

- Prima riga della definizione della funzione main
- main(...) informa che main è una funzione
- int e void sono parole speciali che forniscono informazioni al compilatore
- void: la funzione non ha parametri
- int: la funzione restituisce un intero



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

```
int main(void)
```

```
{
```

```
    printf("from sea  to shining C\n");
```

```
    return 0;
```

```
}
```

- Ogni programma ha una funzione main
- L'esecuzione parte eseguendo questa funzione
- Il corpo della funzione è delimitato dalle {...}
- Le {...} sono usate per raggruppare istruzioni



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

```
int main(void)
```

```
{
```

```
    printf("from sea  to shining C\n");
```

```
    return 0;
```

```
}
```

- Funzione per stampare a schermo (in generale su stdout)
- Funzione contenuta nella standard library
  - un insieme di librerie che forniscono al programmatore funzioni tipizzate
  - gli header file di utilizzare l'insieme di funzioni di libreria associate
  - stdio.h fornisce le informazioni per la printf



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

```
int main(void)
```

```
{
```

```
    printf("from sea to shining C\n");
```

```
    return 0;
```

```
}
```

- Stringa costante
- Argomento della printf
- Controlla cosa viene stampato
- \n è un singolo carattere newline
  - Avanza il cursore all'inizio della riga successiva



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

```
int main(void)
```

```
{
```

```
    printf("from sea to shining C\n");
```

```
    return 0;
```

```
}
```

- Istruzione
- Le istruzioni terminano con un ;
- L'effetto di questa istruzione è stampare a schermo la stringa costante



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


```
int main(void)
```


```
{  
    printf("from sea to shining C\n");
```

```
    return 0;
```

```
}
```

- Istruzione di return
- Ritorna 0 al sistema operativo all'uscita del programma
- Il valore può essere usato per verificare l'uscita del programma
- La } indica la fine della funzione main()

- 
- La funzione printf stampa su tutto lo schermo
    - Da sinistra a destra
    - Dall'alto verso il basso
  - Muove il cursore accapo quando trova il carattere newline
  - Per essere leggibile l'output deve essere opportunamente spaziato sullo schermo



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

```
int main(void)
{
    printf("from sea to ");
    printf("shining ");
    printf("C\n");

    return 0;
}
```






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

```
int main(void)
{
    printf("from sea to ");
    printf("shining ");
    printf("C\n");

    return 0;
}
```

- Programma diverso da quello precedente
- Stampa lo stesso output
- Ogni chiamata della printf inizia da dove la precedente ha lasciato il cursore
- Se vogliamo scrivere su tre linee dobbiamo introdurre dei caratteri newline




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

```
int main(void)
{
    printf("from sea to\n");
    printf("shining\n");
    printf("C\n");

    return 0;
}
```

- In esecuzione stamperà  
from sea to  
shining  
C



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

```
int main(void)
{
    printf("from sea to\n");
    printf("shining\nC\n");

    return 0;
}
```

- Si possono mettere più `\n` nella stessa printf



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

```
int main(void)
```

```
{
```

```
    printf("\n\n\n\n\n\n\n\n\n\n\n");
```

```
    printf("          *****\n");
```

```
    printf("          *   from sea          *\n");
```

```
    printf("          *   to shining C          *\n");
```

```
    printf("          *****\n");
```

```
    printf("\n\n\n\n\n\n\n\n\n\n\n");
```

```
    return 0;
```

```
}
```



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

```
int main(void)
{
    printf("\n\n\n\n\n\n\n\n\n\n");
    printf("          *****\n");
    printf("          * from sea      *\n");
    printf("          * to shining C   *\n");
    printf("          *****\n");
    printf("\n\n\n\n\n\n\n\n\n\n");
    return 0;
}
```

- L'output di questo programma:
  - La stessa frase
  - In una cornice di asterischi

# Variabili, espressioni e assegnamento

- Vediamo un programma che converte una distanza
  - Input distanza maratona in miglia e iarde
  - Output distanza maratona in Km
- Regole
  - 1760 iarde in un miglio
  - 1.609 Km in un miglio



```
/* The distance of a marathon in kilometers. */
```

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

```
int main(void)
```

```
{
```

```
    int    miles, yards;
```

```
    float  kilometers;
```

```
    miles = 26;
```

```
    yards = 385;
```

```
    kilometers = 1.609 * (miles + yards / 1760.0);
```

```
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

```
    return 0;
```

```
}
```

```
/* The distance of a marathon in kilometers. */
```

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

```
int main(void)
```

```
{
```

```
    int    miles, yards;
```

```
    float  kilometers;
```

```
    miles = 26;
```

```
    yards = 385;
```

```
    kilometers = 1.609 * (miles + yards / 1760.0);
```

```
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

```
    return 0;
```

```
}
```

- Compilazione

```
gcc -o marathon marathon.c
```

- L'opzione -o specifica il nome del file di output

- Esecuzione

```
./marathon
```



`/* The distance of a marathon in kilometers. */`

`#include <stdio.h>`

`int main(void)`

`{`

`int miles, yards;`

`float kilometers;`

`miles = 26;`

`yards = 385;`

`kilometers = 1.609 * (miles + yards / 1760.0);`

`printf("\nA marathon is %f kilometers.\n\n", kilometers);`

`return 0;`

`}`

- **Commento**
  - Tutto ciò che è compreso fra `/*` e `*/` viene ignorato dal preprocessore e dal compilatore
  - Anche le linee che iniziano con `//` sono ignorate
- Un commento all'inizio serve a comunicare scopo del programma o delle funzioni
- Si può aggiungere commenti sulle varie versioni e autori

```
/* The distance of a marathon in kilometers. */
```

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

```
int main(void)  
{
```

```
    int    miles, yards;
```

```
    float  kilometers;
```

```
    miles = 26;
```

```
    yards = 385;
```

```
    kilometers = 1.609 * (miles + yards / 1760.0);
```

```
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

```
    return 0;
```

```
}
```

- Dichiarazione

- int è una parola chiave del C

- Indica un tipo di dato fondamentale

- Quelle che seguono sono variabili

- Si indica al compilatore che le variabile sono intere e assumono valori interi.

- Dichiarazioni terminano con un ;

```
/* The distance of a marathon in kilometers. */
```

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

```
int main(void)
```

```
{
```

```
    int    miles, yards;
```

```
    float  kilometers;
```

```
    miles = 26;
```

```
    yards = 385;
```

```
    kilometers = 1.609 * (miles + yards / 1760.0);
```

```
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

```
    return 0;
```

```
}
```

- Dichiarazione

- float è una parola chiave del C
- Indica un tipo di dato fondamentale
- Si indica al compilatore che le variabile sono float e assumono valori reali.
- La variabile kilometers assume valori reali

```
/* The distance of a marathon in kilometers. */
```

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

```
int main(void)
```

```
{
```

```
    int    miles, yards;
```

```
    float  kilometers;
```

```
    miles = 26;
```

```
    yards = 385;
```

```
    kilometers = 1.609 * (miles + yards / 1760.0);
```

```
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

```
    return 0;
```

```
}
```

- Istruzioni di assegnazione
  - = operatore di assegnazione
  - 26 e 385 sono costanti intere
  - 26 assegnato alla variabile miles
  - 285 assegnato alla variabile yards

```
/* The distance of a marathon in kilometers. */
```

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

```
int main(void)
```

```
{
```

```
    int    miles, yards;
```

```
    float  kilometers;
```

```
    miles = 26;
```

```
    yards = 385;
```

```
    kilometers = 1.609 * (miles + yards / 1760.0);
```

```
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

```
    return 0;
```

```
}
```

- Istruzioni di assegnazione
  - Valore assegnato calcolato da una espressione
  - Operazioni dentro le parentesi eseguite per prime
  - Divisione ha precedenza su somma
    - Prima yards/1760.0
    - Il risultato viene sommato a miles e poi moltiplicato per 1.609

```
/* The distance of a marathon in kilometers. */
```

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

```
int main(void)
```

```
{
```

```
    int    miles, yards;
```

```
    float  kilometers;
```

```
    miles = 26;
```

```
    yards = 385;
```

```
    kilometers = 1.609 * (miles + yards / 1760.0);
```

```
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

```
    return 0;
```

```
}
```

- 1.609 e 1760.0 costanti double
- yards/1760.0 diventa una divisione fra double con un risultato double
-

```
/* The distance of a marathon in kilometers. */
```

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

```
int main(void)
```

```
{
```

```
    int    miles, yards;
```

```
    float  kilometers;
```

```
    miles = 26;
```

```
    yards = 385;
```

```
    kilometers = 1.609 * (miles + yards / 1760.0);
```

```
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

```
    return 0;
```

```
}
```

- Chiamata alla funzione printf
- Argomento è la stringa di controllo
- I parametri liberi vengono stampati con la conversione
  - %f specifica che stamperà un float
  - La variabile corrispondente deve essere float
  - In questo caso %f è associato a kilometers

```
/* The distance of a marathon in kilometers. */
```

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

```
int main(void)
```

```
{
```

```
    int    miles, yards;
```

```
    float  kilometers;
```

```
    miles = 26;
```

```
    yards = 385;
```

```
    kilometers = 1.609 * (miles + yards / 1760);
```

```
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

```
    return 0;
```

```
}
```

- Ora 1760 è una costante intera
- yards/1760 divisione fra interi



```
/* The distance of a marathon in kilometers. */
```

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

```
int main(void)
```

```
{
```

```
    int    miles, yards;
```

```
    float  kilometers;
```

```
    miles = 26;
```

```
    yards = 385;
```

```
    kilometers = 1.609 * (miles + yards / 1760);
```

```
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

```
    return 0;
```

```
}
```

- Ora 1760 è una costante intera
- yards/1760 divisione fra interi
- Il resto della divisione si perde e di conseguenza il risultato non è corretto



# Keywords e variabili

- Le parole come `int` e `float` sono keywords
- Le keywords sono riservate
- Non possono essere usate come nomi di variabili
- Anche i nomi delle funzioni non possono essere usati come nomi di variabili
- Non possiamo chiamare una variabile `printf` o `main`


# Uso di #define e #include


- Il compilatore C ha un preprocessore
- Il preprocessore scandisce tutto il file e identifica le direttive per il preprocessore
- Righe che iniziano per #
- #define e #include sono due direttive per il preprocessore

# #define

- Consideriamo le righe di codice

```
#define LIMIT 100
#define PI 3.14159
```
- Il preprocessore sostituirà nel file sorgente
  - Tutte le occorrenze di LIMIT con 100
  - Tutte le occorrenze di PI con 3.14159
- Lascerà solo le occorrenze
  - Tra apici (in stringhe di testo)
  - Nei commenti

- 
- Un `#define` può essere messo ovunque nel file sorgente
  - Avrà effetto solo sulle righe successive
  - Solitamente le `#define` sono messe all'inizio del file
  - Per convenzione si usano le lettere maiuscole nel nome

- 
- L'uso di costanti simboliche
    - Rende il programma più leggibile
    - Rende più facile e sicuro modificarne il valore
      - Ad esempio
        - La velocità della luce  $C$  è approssimativamente 299792.458 Km/s  
`#define C 299792.458`
        - Se un nuovo esperimento producesse una stima più accurata sarebbe facile cambiare in tutto il codice


# #include

- In un programma una riga come


`#include "my_file.h"`

è una direttiva che copia il contenuto di “my\_file.h” in quel punto nel file sorgente

- Una direttiva `#include` può essere messa ovunque
- Tipicamente si mettono all’inizio del file (rende tutto più leggibile)

- 
- Quelli che si includono sono gli header file
  - Per convenzione hanno l'estensione .h
  - Il sistema C fornisce alcuni header file
    - `stdio.h`
    - `stdlib.h`
    - `string.h`
    - `math.h`



- 
- Ogni volta che una funzione di IO deve essere usata (es. `printf`) si deve includere `stdio.h`
  - Il file contiene i prototipi delle funzioni di IO
  - Informazioni necessarie per compilare correttamente il codice sorgente
  - La mancanza non pregiudica la fase di linking

# Un esempio: Pacific Sea

- Pacific sea è il nome dato alla parte di oceano Pacifico visibile dal campus di Santa Cruz sulla baia di Monterey
- Il programma di esempio converte l'area di questa parte di oceano da  $\text{Km}^2$  in varie unità di misura
  - Acri
  - Miglia quadrate
  - Piedi quadrati
  - Pollici quadrati



```
/* Measuring the Pacific Sea. */
```

```
#include "pacific_sea.h"
```

```
int main(void)
```

```
{
```

```
    const int  pacific_sea = AREA;  /* in sq kilometers */  
    double     acres, sq_miles, sq_feet, sq_inches;
```

```
    printf("\nThe Pacific Sea covers an area");  
    printf(" of %d square kilometers.\n", pacific_sea);  
    sq_miles = SQ_MILES_PER_SQ_KILOMETER * pacific_sea;  
    sq_feet = SQ_FEET_PER_SQ_MILE * sq_miles;  
    sq_inches = SQ_INCHES_PER_SQ_FOOT * sq_feet;  
    acres = ACRES_PER_SQ_MILE * sq_miles;  
    printf("In other units of measure this is:\n\n");  
    printf("%22.7e acres\n", acres);  
    printf("%22.7e square miles\n", sq_miles);  
    printf("%22.7e square feet\n", sq_feet);  
    printf("%22.7e square inches\n\n", sq_inches);  
    return 0;
```

```
}
```

```
/* Measuring the Pacific Sea. */
```

```
#include "pacific_sea.h"
```

```
int main(void)
```

```
{
```

```
    const int  pacific_sea = AREA;  /* in sq kilometers */  
    double     acres, sq_miles, sq_feet, sq_inches;
```

```
    printf("\nThe Pacific Sea covers an area");  
    printf(" of %d square kilometers.\n", pacific_sea);  
    sq_miles = SQ_MILES_PER_SQ_KILOMETER * pacific_sea;  
    sq_feet = SQ_FEET_PER_SQ_MILE * sq_miles;  
    sq_inches = SQ_INCHES_PER_SQ_FOOT * sq_feet;  
    acres = ACRES_PER_SQ_MILE * sq_miles;  
    printf("In other units of measure this is:\n\n");  
    printf("%22.7e acres\n", acres);  
    printf("%22.7e square miles\n", sq_miles);  
    printf("%22.7e square feet\n", sq_feet);  
    printf("%22.7e square inches\n\n", sq_inches);  
    return 0;
```

```
}
```

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

```
#define AREA 2337  
#define SQ_MILES_PER_SQ_KILOMETER 0.3861021585424458  
#define SQ_FEET_PER_SQ_MILE (5280 * 5280)  
#define SQ_INCHES_PER_SQ_FOOT 144  
#define ACRES_PER_SQ_MILE 640
```

```
/* Measuring the Pacific Sea. */
```

```
#include "pacific_sea.h"
```

```
int main(void)
{
    const int  pacific_sea = AREA;  /* in sq kilometers */
    double     acres, sq_miles, sq_feet, sq_inches;

    printf("\nThe Pacific Sea covers an area");
    printf(" of %d square kilometers.\n", pacific_sea);
    sq_miles = SQ_MILES_PER_SQ_KILOMETER * pacific_sea;
    sq_feet = SQ_FEET_PER_SQ_MILE * sq_miles;
    sq_inches = SQ_INCHES_PER_SQ_FOOT * sq_feet;
    acres = ACRES_PER_SQ_MILE * sq_miles;
    printf("In other units of measure this is:\n\n");
    printf("%22.7e acres\n", acres);
    printf("%22.7e square miles\n", sq_miles);
    printf("%22.7e square feet\n", sq_feet);
    printf("%22.7e square inches\n\n", sq_inches);
    return 0;
}
```

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

```
#define AREA 2337
#define SQ_MILES_PER_SQ_KILOMETER 0.3861021585424458
#define SQ_FEET_PER_SQ_MILE (5280 * 5280)
#define SQ_INCHES_PER_SQ_FOOT 144
#define ACRES_PER_SQ_MILE 640
```

- 
- La prima include una copia del file `pacic_sea.h`
  - Siccome contiene la direttiva

`#include <stdio.h>`

il preprocessore include anche una copia del file `stdio.h`

- `pacific_sea.h` contiene definizioni utili per il programma
- `stdio.h` incluso perché si fa uso della funzione `printf()`

/\* Measuring the Pacific Sea. \*/

#include "pacific\_sea.h"

int main(void)

{


const int pacific\_sea = AREA; /\* in sq kilometers \*/  
double acres, sq\_miles, sq\_feet, sq\_inches;

printf("\nThe Pacific Sea covers an area");  
printf(" of %d square kilometers.\n", pacific\_sea);  
sq\_miles = SQ\_MILES\_PER\_SQ\_KILOMETER \* pacific\_sea;  
sq\_feet = SQ\_FEET\_PER\_SQ\_MILE \* sq\_miles;  
sq\_inches = SQ\_INCHES\_PER\_SQ\_FOOT \* sq\_feet;  
acres = ACRES\_PER\_SQ\_MILE \* sq\_miles;  
printf("In other units of measure this is:\n\n");  
printf("%22.7e acres\n", acres);  
printf("%22.7e square miles\n", sq\_miles);  
printf("%22.7e square feet\n", sq\_feet);  
printf("%22.7e square inches\n\n", sq\_inches);  
return 0;

}

#include <stdio.h>

#define AREA	2337
#define SQ_MILES_PER_SQ_KILOMETER	0.3861021585424458
#define SQ_FEET_PER_SQ_MILE	(5280 * 5280)
#define SQ_INCHES_PER_SQ_FOOT	144
#define ACRES_PER_SQ_MILE	640

- 
- Definizioni di costanti simboliche
  - Una sola double, le altre intere
  - Tutte rendono il codice più chiaro
  - Se nuova stima dell'area disponibile facile cambiare
  - Le altre rendono chiaro il ruolo dei valori numeri
    - nome significativo alle costanti



```
/* Measuring the Pacific Sea. */
```

```
#include "pacific_sea.h"
```

```
int main(void)
```

```
{
```


```
    const int  pacific_sea = AREA;  /* in sq kilometers */  
    double     acres, sq_miles, sq_feet, sq_inches;
```

```
    printf("\nThe Pacific Sea covers an area");  
    printf(" of %d square kilometers.\n", pacific_sea);  
    sq_miles = SQ_MILES_PER_SQ_KILOMETER * pacific_sea;  
    sq_feet = SQ_FEET_PER_SQ_MILE * sq_miles;  
    sq_inches = SQ_INCHES_PER_SQ_FOOT * sq_feet;  
    acres = ACRES_PER_SQ_MILE * sq_miles;  
    printf("In other units of measure this is:\n\n");  
    printf("%22.7e acres\n", acres);  
    printf("%22.7e square miles\n", sq_miles);  
    printf("%22.7e square feet\n", sq_feet);  
    printf("%22.7e square inches\n\n", sq_inches);  
    return 0;
```

```
}
```

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

```
#define AREA 2337  
#define SQ_MILES_PER_SQ_KILOMETER 0.3861021585424458  
#define SQ_FEET_PER_SQ_MILE (5280 * 5280)  
#define SQ_INCHES_PER_SQ_FOOT 144  
#define ACRES_PER_SQ_MILE 640
```

- 
- Il preprocessore cambia le occorrenze della costante `symbolic` con la stringa di caratteri che la segue
  - E' una operazione
  - Le parentesi non sintatticamente necessarie, ma buona pratica di programmazione
  - Semanticamente più chiara
    - 5280 piedi in un miglio
  - Non si perde efficienza rispetto a scrivere `27878400`
    - Il compilatore espande le espressioni costanti

```
/* Measuring the Pacific Sea. */
```

```
#include "pacific_sea.h"
```

```
int main(void)
```

```
{
```

```
    const int  pacific_sea = AREA;  /* in sq kilometers */
```

```
    double     acres, sq_miles, sq_feet, sq_inches;
```

```
    printf("\nThe Pacific Sea covers an area");
```

```
    printf(" of %d square kilometers.\n", pacific_sea);
```

```
    sq_miles = SQ_MILES_PER_SQ_KILOMETER * pacific_sea;
```

```
    sq_feet = SQ_FEET_PER_SQ_MILE * sq_miles;
```

```
    sq_inches = SQ_INCHES_PER_SQ_FOOT * sq_feet;
```

```
    acres = ACRES_PER_SQ_MILE * sq_miles;
```

```
    printf("In other units of measure this is:\n\n");
```

```
    printf("%22.7e acres\n", acres);
```

```
    printf("%22.7e square miles\n", sq_miles);
```

```
    printf("%22.7e square feet\n", sq_feet);
```

```
    printf("%22.7e square inches\n\n", sq_inches);
```

```
    return 0;
```

```
}
```

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


```
#define AREA 2337
```

```
#define SQ_MILES_PER_SQ_KILOMETER 0.3861021585424458
```

```
#define SQ_FEET_PER_SQ_MILE (5280 * 5280)
```

```
#define SQ_INCHES_PER_SQ_FOOT 144
```

```
#define ACRES_PER_SQ_MILE 640
```

- 
- Prima della compilazione AREA è sostituito da 2337
  - Il compilatore interpreta la linea di codice come
    - Dichiarazione di una variabile `pacific_sea`
    - La variabile è di tipo `int`
    - La keyword `const` qualifica il tipo
      - La variabile può essere inizializzata
      - Il suo valore non può essere modificato

```
/* Measuring the Pacific Sea. */
```

```
#include "pacific_sea.h"
```

```
int main(void)
```

```
{
```

```
    const int  pacific_sea = AREA;  /* in sq kilometers */
```

```
    double     acres, sq_miles, sq_feet, sq_inches;
```

```
    printf("\nThe Pacific Sea covers an area");
```

```
    printf(" of %d square kilometers.\n", pacific_sea);
```

```
    sq_miles = SQ_MILES_PER_SQ_KILOMETER * pacific_sea;
```

```
    sq_feet = SQ_FEET_PER_SQ_MILE * sq_miles;
```

```
    sq_inches = SQ_INCHES_PER_SQ_FOOT * sq_feet;
```

```
    acres = ACRES_PER_SQ_MILE * sq_miles;
```

```
    printf("In other units of measure this is:\n\n");
```

```
    printf("%22.7e acres\n", acres);
```

```
    printf("%22.7e square miles\n", sq_miles);
```

```
    printf("%22.7e square feet\n", sq_feet);
```

```
    printf("%22.7e square inches\n\n", sq_inches);
```

```
    return 0;
```

```
}
```

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


```
#define AREA 2337
```

```
#define SQ_MILES_PER_SQ_KILOMETER 0.3861021585424458
```

```
#define SQ_FEET_PER_SQ_MILE (5280 * 5280)
```

```
#define SQ_INCHES_PER_SQ_FOOT 144
```

```
#define ACRES_PER_SQ_MILE 640
```

- 
- Dichiarazione di variabili di tipo double
  - In C i tipi floating (reali) sono
    - float (6 cifre significative)
    - double (15 cifre significative)
    - long double (15 cifre significative)
    - Valori tipici, ma dipendenti dalla implementazione

/\* Measuring the Pacific Sea. \*/

#include "pacific\_sea.h"

int main(void)

{


const int pacific\_sea = AREA; /\* in sq kilometers \*/  
double acres, sq\_miles, sq\_feet, sq\_inches;

printf("\nThe Pacific Sea covers an area");  
printf(" of %d square kilometers.\n", pacific\_sea);  
sq\_miles = SQ\_MILES\_PER\_SQ\_KILOMETER \* pacific\_sea;  
sq\_feet = SQ\_FEET\_PER\_SQ\_MILE \* sq\_miles;  
sq\_inches = SQ\_INCHES\_PER\_SQ\_FOOT \* sq\_feet;  
acres = ACRES\_PER\_SQ\_MILE \* sq\_miles;  
printf("In other units of measure this is:\n\n");  
printf("%22.7e acres\n", acres);  
printf("%22.7e square miles\n", sq\_miles);  
printf("%22.7e square feet\n", sq\_feet);  
printf("%22.7e square inches\n\n", sq\_inches);  
return 0;

}

#include <stdio.h>


#define AREA	2337
#define SQ_MILES_PER_SQ_KILOMETER	0.3861021585424458
#define SQ_FEET_PER_SQ_MILE	(5280 * 5280)
#define SQ_INCHES_PER_SQ_FOOT	144
#define ACRES_PER_SQ_MILE	640


- 
- L'istruzione stampa la linea  
5.7748528e+05 acres
  - Il numero è scritto in notazione scientifica e significa  $5.7748528 * 10^5$
  - La conversione %e indica che si vuole scrivere in notazione scientifica (detta e-format)
  - Il formato %m.ne (m e n interi positivi) indica che
    - m spazi in totale
    - n cifre dopo la virgola
    - In questo caso 22 spazi con 7 cifre dopo la virgola





# Uso di printf() e scanf()

- printf() si usa per l'output
- scanf() si usa per l'input
- f sta per formatted
- Il primo parametro è appunto la stringa di controllo che stabilisce il formato
- Non fanno parte del linguaggio, ma del sistema C
- Contenuta nella standard library
- Linkata alla fine della compilazione implicitamente

- 
- Il programmatore deve comunque dichiarare al compilatore le funzioni che usa
  - Il meccanismo nell'ANSI C è quello del prototipo
  - I prototipi delle funzioni della standard library sono negli standard header file.
  - In particolare per `printf()` e `scanf()` in `stdio.h`


- 
- Le due funzioni accettano una lista di argomenti
    - Stringa di controllo
    - Altri argomenti legati alla stringa di controllo
  - La stringa di controllo
    - È una stringa
    - Contiene specifiche di conversione
    - Una conversione è identificata dal carattere % seguita da un carattere di conversione (es. %d per intero)
  - Altri argomenti sono uno per ogni conversione nella stringa di controllo.


- 
- Per stampare le prime 3 lettere dell'alfabeto
    - Senza conversione  
`printf("abc");`
    - Conversione per char  
`printf("%c%c%c",'a','b','c');`
    - Conversione per stringa  
`printf("%s","abc");`

- 
- Si può decidere quanto spazio dare alle conversioni  
`printf("%c%3c%5c\n",'A','B','C');`
  - `%mc` significa quanti caratteri sono usati per quella conversione
  - Allineamento a destra
  - Output  
A B C

# Caratteri di conversione printf()

- %c: carattere
- %d: intero decimale
- %e: reale in notazione scientifica
- %f: reale
- %g: il più corto fra %e e %f
- %s: stringa


- 
- La scanf() è analoga alla printf()
  - Usata per l'input
  - Primo parametro è la stringa di controllo
  - Input è sempre una sequenza di caratteri
  - Conversioni specificano come i vari caratteri dell'input devono essere interpretati

- 
- I rimanenti caratteri sono indirizzi dove scrivere i vari input
    - `scanf(“%d”,&x);`
    - `&` operatore di indirizzo
    - `&x`: indirizzo di `x`



# Caratteri di conversione scanf()

- %c: carattere
- %d: intero decimale
- %f: float
- %lf: double
- %s: stringa



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

```
int main(void)
```

```
{
```

```
    char    c1, c2, c3;
```

```
    int     i;
```

```
    float   x;
```

```
    double  y;
```

```
    printf("\n%s\n%s", "Input three characters ", "an int, a float, and a double ");
```

```
    scanf("%c%c%c%d%f%lf", &c1, &c2, &c3, &i, &x, &y);
```

```
    printf("\nHere is the data that you typed in:\n");
```

```
    printf("%3c%3c%3c%5d%17e%17e\n\n", c1, c2, c3, i, x, y);
```

```
    return 0;
```


```
}
```

Input three characters,

an int, a float, and a double: ABC 3 55 77.7

Here is the data that you typed in:

A B C 3 5.500000e+01 7.770000e+01



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

```
int main(void)
```

```
{
```

```
    char    c1, c2, c3;
```

```
    int     i;
```

```
    float   x;
```

```
    double  y;
```

```
    printf("\n%s\n%s", "Input three characters ", "an int, a float, and a double ");
```

```
    scanf("%c%c%c%d%f%lf", &c1, &c2, &c3, &i, &x, &y);
```

```
    printf("\nHere is the data that you typed in:\n");
```

```
    printf("%3c%3c%3c%5d%17e%17e\n\n", c1, c2, c3, i, x, y);
```

```
    return 0;
```


```
}
```

Input three characters,

an int, a float, and a double: AB C 3 55 77.7

Here is the data that you typed in:


A B -615754144 3.076130e-41 6.953071e-310

- 
- La scanf
    - Quando legge numeri ignora gli spazi
    - Non li ignora quando legge caratteri
    - Lo spazio bianco è un carattere
  - Quindi la lettera C viene letta come un intero



# Flusso di controllo


- Le istruzioni sono normalmente eseguite in sequenza
- La maggior parte degli algoritmi richiedono di alterare il flusso di controllo sequenziale
- Due costrutti
  - if e if-else fornisce la scelta di azioni alternative
  - while e for forniscono la possibilità di iterare

- 
- Richiedono di valutare delle espressioni logiche
  - Espressioni che possono essere TRUE o FALSE
  - In C FALSE è rappresentato dal valore 0
  - Ogni valore non-zero è interpretato come TRUE

# if


- La forma generale è

```
if (expr) {  
    istruzioni-if;  
}  
istruzione successiva;
```
- Se `expr` è non-zero (TRUE) si esegue `istruzioni-if` e poi `istruzione successiva`
- Se `expr` è 0 (FALSE) si esegue direttamente `istruzione successiva`




```
a = 1;  
if (b == 3) {  
    a = 5;  
}  
printf("%d", a);
```





```
a = 1;
if (b == 3) {
    a = 5;
}
printf("%d", a);
```

- Il simbolo == è l'operatore is equal. Verifica l'eguaglianza
- Da non confondere con l'operatore di assegnazione =
- Si verifica se la variabile b è uguale a 3
  - Se vero si assegna il valore 5 alla variabile a
  - Se falso si esegue subito la stampa di a
- Effetti visibili
  - Se vero viene stampato il valore 5
  - Se falso viene stampato il valore 1




```
a = 1;
if (b == 3) {
    a = 5;
    c = 7;
}
printf("%d", a);
```

- Le istruzioni all'interno dell'if possono anche essere una sequenza
- In tal caso devo essere necessariamente fra { }
- Suggerimento:
  - Usare sempre le parentesi, anche per una sola istruzione
  - Evita errori nel caso si dovesse espandere il blocco


# if-else

- La forma generale è

```
if (expr) {  
    istruzioni-if  
} else {  
    istruzioni-else  
}
```
- Se `expr` è non-zero (TRUE) si esegue il primo blocco di istruzioni `istruzioni-if`
- Se `expr` è zero (FALSE) si esegue il secondo blocco di istruzioni `istruzioni-else`




```
if (cnt == 0) {  
    a = 2;  
    b = 3;  
    c = 5;  
} else {  
    a = -1;  
    b = -2;  
    c = -3;  
}  
printf("%d", a+b+c);
```




```
if (cnt == 0) {  
    a = 2;  
    b = 3;  
    c = 5;  
} else {  
    a = -1;  
    b = -2;  
    c = -3;  
}  
printf("%d", a+b+c);
```

- Se la variabile cnt ha il valore 0 il programma stampa 10
- Se la variabile cnt non ha valore 0 viene stampa -6




```
if (cnt) {  
    a = 2;  
    b = 3;  
    c = 5;  
} else {  
    a = -1;  
    b = -2;  
    c = -3;  
}  
printf("%d", a+b+c);
```



```
if (cnt) {  
    a = 2;  
    b = 3;  
    c = 5;  
} else {  
    a = -1;  
    b = -2;  
    c = -3;  
}  
printf("%d", a+b+c);
```


- Se la variabile cnt ha il valore 0 il programma stampa -6
- Se la variabile cnt non ha valore 0 viene stampa 10



```
if (cnt) {  
    a = 2;  
    b = 3;  
    c = 5;  
} else {  
    a = -1;  
    b = -2;  
    c = -3;  
}  
printf("%d", a+b+c);
```

- Se la variabile cnt ha il valore 0 il programma stampa -6
- Se la variabile cnt non ha valore 0 viene stampa 10
- In questo caso non c'è istruzione di confronto
- La decisione è basata solo sul valore di cnt
  - Se cnt ha valore 0 allora FALSE
  - Se cnt ha valore non-zero allora TRUE





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

```
int main(void)
```

```
{
```

```
    int i = 1, sum = 0;
```

```
    while (i <= 5) {
```

```
        sum += i;
```


```
        ++i;
```

```
    }
```

```
    printf("sum = %d\n", sum);
```

```
    return 0;
```

```
}
```



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

```
int main(void)
```

```
{
```

```
    int i = 1, sum = 0;
```

```
    while (i <= 5) {
```

```
        sum += i;
```

```
        ++i;
```


```
    }
```

```
    printf("sum = %d\n", sum);
```

```
    return 0;
```

```
}
```

- Ciclo while
- $\leq$  significa minore o uguale: verifica se i è minore uguale a 5
- Se la condizione è verificata si eseguono le istruzioni del blocco fino alla }
- Il controllo ritorna all'inizio del ciclo while per ripetere tutto il processo
- L'iterazione continua fino a quando il test non fallisce



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

```
int main(void)
```

```
{
```

```
    int i = 1, sum = 0;
```

```
    while (i <= 5) {
```

```
        sum += i;
```

```
        ++i;
```


```
    }
```

```
    printf("sum = %d\n", sum);
```

```
    return 0;
```

```
}
```

- Istruzione di assegnazione
- Equivalente a  $\text{sum} = \text{sum} + i$ ;
- Somma a sum il valore di i e assegna il risultato a sum



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

```
int main(void)
```

```
{
```

```
    int i = 1, sum = 0;
```

```
    while (i <= 5) {
```

```
        sum += i;
```

```
        ++i;
```

```
    }
```

```
    printf("sum = %d\n", sum);
```

```
    return 0;
```


```
}
```

- Istruzione di assegnazione
- Equivalente a  $sum = sum + i$ ;
- Somma a sum il valore di i e assegna il risultato a sum
- In generale una istruzione del tipo

variabile *op*= expr; (*op* = +, \*, -, /)

è equivalente a

variabile = variabile *op* expr;



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

```
int main(void)
```

```
{
```

```
    int i = 1, sum = 0;
```

```
    while (i <= 5) {
```

```
        sum += i;
```

```
        ++i;
```


```
    }
```

```
    printf("sum = %d\n", sum);
```

```
    return 0;
```

```
}
```

- Operatore di incremento
- Equivalente a  $i = i + 1$ ;
- Analogamente
  - $i++$       equivalente a  $i = i + 1$
  - $--i$       equivalente a  $i = i - 1$
  - $i--$       equivalente a  $i = i - 1$
- Attenzione al loro uso all'interno di espressioni: il loro funzionamento è più complicato.



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

```
int main(void)
```

```
{
```

```
    int i = 1, sum = 0;
```

```
    while (i <= 5) {
```

```
        sum += i;
```

```
        ++i;
```


```
    }
```

```
    printf("sum = %d\n", sum);
```

```
    return 0;
```

```
}
```

- Funzione di stampa a schermo
- Il programma stamperà



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

```
int main(void)
```

```
{
```

```
    int i = 1, sum = 0;
```

```
    while (i <= 5) {
```

```
        sum += i;
```

```
        ++i;
```

```
    }
```

```
    printf("sum = %d\n", sum);
```

```
    return 0;
```

```
}
```

- Funzione di stampa a schermo
- Il programma stamperà

sum = 15

# while

- La forma generale del while è

```
while (expr) {  
    istruzioni_while;  
}  
istruzioni;
```
- Se `expr` non-zero (TRUE) si esegue `istruzioni_while` e poi si verifica nuovamente `expr`
- Il processo continue fino a quando `expr` è 0.



# while

- La forma generale del while è

```
while (expr) {  
    istruzioni_while;  
}  
istruzioni;
```
- Se `expr` non-zero (TRUE) si esegue `istruzioni_while` e poi si verifica nuovamente `expr`
- Il processo continue fino a quando `expr` è 0.
- Necessaria una istruzione nel ciclo che possa modificare il valore di `expr` (`++i` nell'esempio)

# for

- Forma generale

```
for (expr1; expr2; expr3) {  
    istruzioni_for;  
}
```

- Equivalente a

```
expr1;  
while (expr2) {  
    istruzioni_for;  
    expr3;  
}
```

- Tipicamente
  - Expr1 esegue un'assegnazione iniziale
  - Expr2 esegue un test
  - Expr3 incrementa una variabile
- Ad esempio

```
for (i=1; i<=5; i++) {  
    sum += i;  
}
```
- Equivalente al while nell'esempio precedente



```
/* Compute the minimum, maximum, sum, and average. */
```

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

```
#include <stdlib.h>
```

```
int main(void)
```

```
{
```

```
    int    i;
```

```
    double x, min, max, sum, avg;
```

```
    if (scanf("%lf", &x) != 1) {
```

```
        printf("No data found - bye!\n");
```

```
        exit(1);
```

```
    }
```

```
    min = max = sum = avg = x;
```

```
    printf("%5s%9s%9s%9s%12s%12s\n%5s%9s%9s%9s%12s%12s\n\n",
```

```
        "Count", "Item", "Min", "Max", "Sum", "Average",
```

```
        "-----", "-----", "----", "----", "----", "-----");
```

```
    printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",
```

```
        1, x, min, max, sum, avg);
```

```
    for (i = 2; scanf("%lf", &x) == 1; ++i) {
```

```
        if (x < min) {
```

```
            min = x;
```

```
        }
```

```
        else if (x > max) {
```

```
            max = x;
```

```
        }
```

```
        sum += x;
```

```
        avg = sum / i;
```


```
        printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",
```

```
            i, x, min, max, sum, avg);
```

```
    }
```

```
    return 0;
```

```
}
```

- 
- Una volta compilato
    - `Gcc -o running_sum running_sum.c`
  - Si manda in esecuzione  
`./running_sum`



## Input

3    -5    7    -9    11    -13    15    -17    19    -21

## Output

Count	Item	Min	Max	Sum	Average
-----	-----	---	---	---	-----
1	3.0	3.0	3.0	3.000	3.000
2	-5.0	-5.0	3.0	-2.000	-1.000
3	7.0	-5.0	7.0	5.000	1.667
4	-9.0	-9.0	7.0	-4.000	-1.000
5	11.0	-9.0	11.0	7.000	1.400
6	-13.0	-13.0	11.0	-6.000	-1.000
7	15.0	-13.0	15.0	9.000	1.286
8	-17.0	-17.0	15.0	-8.000	-1.000
9	19.0	-17.0	19.0	11.000	1.222
10	-21.0	-21.0	19.0	-10.000	-1.000



## Input

3   -5   7   -9   11   -13   15   -17   19   -21

## Output

Count	Item	Min	Max	Sum	Average
-----	-----	---	---	---	-----
1	3.0	3.0	3.0	3.000	3.000
2	-5.0	-5.0	3.0	-2.000	-1.000
3	7.0	-5.0	7.0	5.000	1.667
4	-9.0	-9.0	7.0	-4.000	-1.000
5	11.0	-9.0	11.0	7.000	1.400
6	-13.0	-13.0	11.0	-6.000	-1.000
7	15.0	-13.0	15.0	9.000	1.286
8	-17.0	-17.0	15.0	-8.000	-1.000
9	19.0	-17.0	19.0	11.000	1.222
10	-21.0	-21.0	19.0	-10.000	-1.000

Diverse maniere di dare l'input

- uno alla volta
- in una riga
- file



```
/* Compute the minimum, maximum, sum, and average. */
```

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

```
#include <stdlib.h>
```

```
int main(void)
```

```
{
```

```
    int    i;
```

```
    double x, min, max, sum, avg;
```

```
    if (scanf("%lf", &x) != 1) {  
        printf("No data found - bye!\n");  
        exit(1);  
    }
```

```
    min = max = sum = avg = x;
```

```
    printf("%5s%9s%9s%9s%12s%12s\n%5s%9s%9s%9s%12s%12s\n\n",  
        "Count", "Item", "Min", "Max", "Sum", "Average",  
        "-----", "-----", "----", "----", "----", "-----");
```

```
    printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",  
        1, x, min, max, sum, avg);
```

```
    for (i = 2; scanf("%lf", &x) == 1; ++i) {
```

```
        if (x < min) {
```

```
            min = x;
```

```
        }
```

```
        else if (x > max) {
```

```
            max = x;
```

```
        }
```

```
        sum += x;
```

```
        avg = sum / i;
```


```
        printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",  
            i, x, min, max, sum, avg);
```

```
    }
```

```
    return 0;
```

```
}
```



- 
- La scanf ritorna il numero di conversioni che hanno avuto successo
  - Una sola conversione. Diverso da 1 non ha letto niente.
  - Exit(), prototipo in stdlib.h, il programma è terminato
  - Se non riusciamo a leggere un primo dato usciamo con un messaggio di errore.

/\* Compute the minimum, maximum, sum, and average. \*/

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

int main(void)

{

int i;

double x, min, max, sum, avg;

```
if (scanf("%lf", &x) != 1) {  
    printf("No data found - bye!\n");  
    exit(1);  
}
```

min = max = sum = avg = x;

```
printf("%5s%9s%9s%9s%12s%12s\n%5s%9s%9s%9s%12s%12s\n\n",  
       "Count", "Item", "Min", "Max", "Sum", "Average",  
       "-----", "-----", "-----", "-----", "-----", "-----");
```

```
printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",
```

1, x, min, max, sum, avg);

for (i = 2; scanf("%lf", &amp;x) == 1; ++i) {

if (x &lt; min) {

min = x;

}

else if (x &gt; max) {

max = x;

}

sum += x;

avg = sum / i;

```
printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",
```

i, x, min, max, sum, avg);

}

return 0;

}



## Input

3    -5    7    -9    11    -13    15    -17    19    -21

## Output

Count	Item	Min	Max	Sum	Average
-----	-----	---	---	---	-----
1	3.0	3.0	3.0	3.000	3.000
2	-5.0	-5.0	3.0	-2.000	-1.000
3	7.0	-5.0	7.0	5.000	1.667
4	-9.0	-9.0	7.0	-4.000	-1.000
5	11.0	-9.0	11.0	7.000	1.400
6	-13.0	-13.0	11.0	-6.000	-1.000
7	15.0	-13.0	15.0	9.000	1.286
8	-17.0	-17.0	15.0	-8.000	-1.000
9	19.0	-17.0	19.0	11.000	1.222
10	-21.0	-21.0	19.0	-10.000	-1.000

Caso in cui è utile avere stringhe costanti come parametri



```
/* Compute the minimum, maximum, sum, and average. */
```

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

```
#include <stdlib.h>
```

```
int main(void)
```

```
{
```

```
    int    i;
```

```
    double x, min, max, sum, avg;
```

```
    if (scanf("%lf", &x) != 1) {  
        printf("No data found - bye!\n");  
        exit(1);  
    }
```

```
    min = max = sum = avg = x;
```

```
    printf("%5s%9s%9s%9s%12s%12s\n%5s%9s%9s%9s%12s%12s\n\n",  
        "Count", "Item", "Min", "Max", "Sum", "Average",  
        "-----", "-----", "----", "----", "----", "-----");
```

```
    printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",  
        1, x, min, max, sum, avg);
```

```
    for (i = 2; scanf("%lf", &x) == 1; ++i) {  
        if (x < min) {
```

```
            min = x;
```

```
        }
```

```
        else if (x > max) {
```

```
            max = x;
```

```
        }
```

```
        sum += x;
```

```
        avg = sum / i;
```

```
        printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",  
            i, x, min, max, sum, avg);
```

```
    }
```

```
    return 0;
```

```
}
```



**Input**

3    -5    7    -9    11    -13    15    -17    19    -21

**Output**

Count	Item	Min	Max	Sum	Average
-----	-----	---	---	---	-----
1	3.0	3.0	3.0	3.000	3.000
2	-5.0	-5.0	3.0	-2.000	-1.000
3	7.0	-5.0	7.0	5.000	1.667
4	-9.0	-9.0	7.0	-4.000	-1.000
5	11.0	-9.0	11.0	7.000	1.400
6	-13.0	-13.0	11.0	-6.000	-1.000
7	15.0	-13.0	15.0	9.000	1.286
8	-17.0	-17.0	15.0	-8.000	-1.000
9	19.0	-17.0	19.0	11.000	1.222
10	-21.0	-21.0	19.0	-10.000	-1.000



```
/* Compute the minimum, maximum, sum, and average. */
```

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

```
#include <stdlib.h>
```

```
int main(void)
```

```
{
```

```
    int    i;
```

```
    double  x, min, max, sum, avg;
```

```
    if (scanf("%lf", &x) != 1) {  
        printf("No data found - bye!\n");  
        exit(1);  
    }
```

```
    min = max = sum = avg = x;
```

```
    printf("%5s%9s%9s%9s%12s%12s\n%5s%9s%9s%9s%12s%12s\n\n",  
           "Count", "Item", "Min", "Max", "Sum", "Average",  
           "-----", "-----", "----", "----", "----", "-----");
```

```
    printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",  
           1, x, min, max, sum, avg);
```

```
    for (i = 2; scanf("%lf", &x) == 1; ++i) {
```

```
        if (x < min) {
```

```
            min = x;
```

```
        }
```

```
        else if (x > max) {
```

```
            max = x;
```

```
        }
```

```
        sum += x;
```


```
        avg = sum / i;
```

```
        printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",  
               i, x, min, max, sum, avg);
```

```
    }
```

```
    return 0;
```

```
}
```

- 
- Inizialmente viene assegnato  $i=2$
  - Poi si testa la condizione `scanf("%lf", &x) == 1`
  - La conversione ha successo se `scanf()`
    - legge correttamente dallo standard input
    - interpreta ciò che legge come un double
    - conserva il valore all'indirizzo della variabile `x`
  - Il loop è eseguito fino a quando
  - La variabile `i` è incrementata alla fine di ogni ciclo



```
/* Compute the minimum, maximum, sum, and average. */
```

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

```
#include <stdlib.h>
```

```
int main(void)
```

```
{
```

```
    int    i;
```

```
    double x, min, max, sum, avg;
```

```
    if (scanf("%lf", &x) != 1) {  
        printf("No data found - bye!\n");  
        exit(1);  
    }
```

```
    min = max = sum = avg = x;
```

```
    printf("%5s%9s%9s%9s%12s%12s\n%5s%9s%9s%9s%12s%12s\n\n",  
        "Count", "Item", "Min", "Max", "Sum", "Average",  
        "-----", "-----", "----", "----", "----", "-----");
```

```
    printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",  
        1, x, min, max, sum, avg);
```

```
    for (i = 2; scanf("%lf", &x) == 1; ++i) {
```

```
        if (x < min) {
```

```
            min = x;
```

```
        }
```

```
        else if (x > max) {
```

```
            max = x;
```

```
        }
```

```
        sum += x;
```

```
        avg = sum / i;
```


```
        printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",  
            i, x, min, max, sum, avg);
```


```
    }
```


```
    return 0;
```


```
}
```



- 
- Istruzione if-else
  - L'istruzione else è a sua volta un if
  - L'effetto è di aggiornare, se necessario, il max o il min

- 
- Istruzione if-else
  - L'istruzione else è a sua volta un if
  - L'effetto è di aggiornare, se necessario, il max o il min
  - Si poteva scrivere
    - if (x < min) min = x;
    - if (x > max) max = x;

- 
- Nel programma abbiamo usato un ciclo for
  - Viene fatta una prima lettura prima del ciclo

- 
- Nel programma abbiamo usato un ciclo for
    - Classicamente quando è noto il numero di iterazioni
    - `for (i=1; i<N; i++)`
  - Viene fatta una prima lettura prima del ciclo
    - Quando possibile più elegante mettere tutto all'interno del ciclo

/\* Compute the minimum, maximum, sum, and average. \*/

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

int i=0;

double x, min, max, sum, avg;

while (scanf("%lf", &x) == 1) {

i = i + 1;

if (i==1) {

sum = avg = min = max = x;

printf("%5s%9s%9s%9s%12s%12s\n%5s%9s%9s%9s%12s%12s\n\n",

"Count", "Item", "Min", "Max", "Sum", "Average",

"-----", "-----", "----", "----", "----", "-----");

} else {

if (x<min) min = x;

else if (x > max) max = x;

sum += x;

avg = sum/i;

}

printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",

i, x, min, max, sum, avg);

}

if (i==0) {

printf("No data found - bye!\n");

}

return 0;

}

```
/* Compute the minimum, maximum, sum, and average. */
```

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

```
#include <stdlib.h>
```

```
int main(void)
```

```
{
```

```
    int    i=0;
```

```
    double x, min, max, sum, avg;
```

```
    while (scanf("%lf", &x) == 1) {
```

```
        i = i + 1;
```

```
        if (i==1) {
```

```
            sum = avg = min = max = x;
```

```
            printf("%5s%9s%9s%9s%12s%12s\n%5s%9s%9s%9s%12s%12s\n\n",
```

```
                "Count", "Item", "Min", "Max", "Sum", "Average",
```

```
                "-----", "-----", "----", "----", "----", "-----");
```

```
        } else {
```

```
            if (x<min) min = x;
```

```
            else if (x > max) max = x;
```

```
            sum += x;
```

```
            avg = sum/i;
```

```
        }
```

```
        printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",
```

```
            i, x, min, max, sum, avg);
```

```
    }
```


```
    if (i==0) {
```

```
        printf("No data found - bye!\n");
```

```
    }
```

```
    return 0;
```

```
}
```

- 
- Lettura solo nel ciclo while
  - Controllo sull'esistenza dell'input alla fine
  - Solitamente i variabile di iterazione. Qui conta quanti elementi in input

/\* Compute the minimum, maximum, sum, and average. \*/

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

```
int main(void)
{
    int    cont=0;
    double  x, min, max, sum, avg;

    while (scanf("%lf", &x) == 1) {
        cont = cont + 1;
        if (cont==1) {
            sum = avg = min = max = x;
            printf("%5s%9s%9s%9s%12s%12s\n%5s%9s%9s%9s%12s%12s\n\n",
                "Count", "Item", "Min", "Max", "Sum", "Average",
                "-----", "-----", "----", "----", "----", "-----");
        } else {
            if (x<min) min = x;
            else if (x > max) max = x;
            sum += x;
            avg = sum/cont;
        }
        printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",
            cont, x, min, max, sum, avg);
    }
    if (cont==0) {
        printf("No data found - bye!\n");
    }

    return 0;
}
```





# Problemi di compilazione

- I messaggi del compilatore sono importanti e vanno letti
- A volte però possono creare confusione
- Proviamo a compilare il file seguente  
(running\_sum2Errore.c)


\* Compute the minimum, maximum, sum, and average. \*/

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

```
int main(void)
{
    int    cont=0;
    double  x, min, max, sum, avg;

    while (scanf("%lf", &x) == 1) {
        cont = cont + 1;
        if (cont==1) {
            sum = avg = min = max = x;
            printf("%5s%9s%9s%9s%12s%12s\n%5s%9s%9s%9s%12s%12s\n\n",
                "Count", "Item", "Min", "Max", "Sum", "Average",
                "-----", "-----", "----", "----", "----", "-----");
        } else {
            if (x<min) min = x;
            else if (x > max) max = x;
            sum += x;
            avg = sum/cont;
        }
        printf("%5d%9.1f%9.1f%9.1f%12.3f%12.3f\n",
            cont, x, min, max, sum, avg);
    }
    if (cont==0) {
        printf("No data found - bye!\n");
    }

    return 0;
}
```

- 
- Lunga deprimente lista di errori
  - Errori anche negli header file di sistema!!!!
  - Primo errore

```
running_sum2Errore.c:1:11: error:  
expected '=', ',', ';', 'asm' or  
'__attribute__' before 'the'
```


```
* Compute the minimum, maximum, sum,  
and average. */
```


- Errore alla riga 1




# Funzioni

- Una funzione è un pezzo di codice che rappresenta un blocco del processo di problem solving
- Un programma C è costituito da una o più funzioni su uno o più file sorgenti
- Una delle funzioni è necessariamente la funzione `main()` da dove inizia l'esecuzione del programma
- Le altre funzioni sono richiamate all'interno della `main()` e dall'interno di altre funzioni.

- 
- Le funzioni devono essere dichiarate prima di usarle
  - Vogliamo usare la funzione `pow()` (elevamento a potenza) disponibile nella libreria matematica
    - La chiamata `pow(x,y)` ritorna  $x^y$
  - La dichiarazione della funzione è data da
    - `double pow(double x, double y);`
  - Questo tipo di dichiarazione si chiama *prototipo della funzione*

- 
- I nomi dei parametri nel prototipo sono ignorati dal compilatore
  - E' equivalente il prototipo
    - `double pow(double, double );`
  - Al compilatore interessa solo il tipo dei parametri e il tipo della funzione
  - Buona norma mettere comunque i parametri, specialmente se hanno nomi significativi

- 
- La forma generale di un prototipo è  
*type function\_name(parameter type list);*
  - La parola chiave void si usa nel caso
    - La funzione non ha parametri  
`int f(void)`
    - La funzione non ritorna niente  
`void f(int)`
  - Gli argomenti se necessario sono convertiti al tipo specificato nel prototipo

# Esempio: maxmin

- Un programma che
  - Stampi informazioni sul programma
  - Legga un valore intero  $n$
  - Legga  $n$  numeri reali
  - Trovi il minima e il massimo valore fra i numeri reali



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

```
float maximum(float x, float y);
```

```
float minimum(float x, float y);
```

```
void prn_info(void);
```

```
int main(void)
```

```
{
```

```
    int i, n;
```

```
    float max, min, x;
```

```
    prn_info();
```

```
    printf("Input n: ");
```

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

```
    printf("\nInput %d real numbers: ", n);
```

```
    scanf("%f", &x);
```

```
    max = min = x;
```

```
    for (i = 2; i <= n; ++i) {
```

```
        scanf("%f", &x);
```

```
        max = maximum(max, x);
```

```
        min = minimum(min, x);
```

```
    }
```

```
    printf("\n%s%11.3f\n%s%11.3f\n\n",
```

```
        "Maximum value:", max,
```

```
        "Minimum value:", min);
```

```
    return 0;
```

```
}
```

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

```
float  maximum(float x, float y);  
float  minimum(float x, float y);  
void   prn_info(void);
```

```
int main(void)  
{  
    int    i, n;  
    float  max, min, x;  
  
    prn_info();  
    printf("Input n: ");  
    scanf("%d", &n);  
    printf("\nInput %d real numbers: ", n);  
    scanf("%f", &x);  
    max = min = x;  
    for (i = 2; i <= n; ++i) {  
        scanf("%f", &x);  
        max = maximum(max, x);  
        min = minimum(min, x);  
    }  
    printf("\n%s%11.3f\n%s%11.3f\n\n",  
        "Maximum value:", max,  
        "Minimum value:", min);  
    return 0;  
}
```

```
float maximum(float x, float y)  
{  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

```
float minimum(float x, float y)  
{  
    if (x < y)  
        return x;  
    else  
        return y;  
}
```

```
void prn_info(void)  
{  
    printf("\n%s\n%s\n\n",  
        "This program reads an integer value  
for n, and then",  
        "processes n real numbers to find max  
and min values.");  
}
```

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


```
float  maximum(float x, float y);  
float  minimum(float x, float y);  
void   prn_info(void);
```

```
int main(void)  
{  
    int    i, n;  
    float  max, min, x;  
  
    prn_info();  
    printf("Input n: ");  
    scanf("%d", &n);  
    printf("\nInput %d real numbers: ", n);  
    scanf("%f", &x);  
    max = min = x;  
    for (i = 2; i <= n; ++i) {  
        scanf("%f", &x);  
        max = maximum(max, x);  
        min = minimum(min, x);  
    }  
    printf("\n%s%11.3f\n%s%11.3f\n\n",  
        "Maximum value:", max,  
        "Minimum value:", min);  
    return 0;  
}
```

```
float maximum(float x, float y)  
{  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

```
float minimum(float x, float y)  
{  
    if (x < y)  
        return x;  
    else  
        return y;  
}
```

```
void prn_info(void)  
{  
    printf("\n%s\n%s\n\n",  
        "This program reads an integer value  
for n, and then",  
        "processes n real numbers to find max  
and min values.");  
}
```

- 
- I prototipi per le funzioni `maximum()`, `minimum()` e `prn_info()`
  - Posti all'inizio del file
  - I primi due informano che le funzioni
    - Hanno due argomenti di tipo `float`
    - Ritornano un valore di tipo `float`
  - La funzione `prn_info()`
    - Non ha argomenti
    - Non ritorna alcun valore

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

```
float  maximum(float x, float y);  
float  minimum(float x, float y);  
void   prn_info(void);
```

```
int main(void)  
{
```

```
    int    i, n;  
    float  max, min, x;
```

```
    prn_info();
```

```
    printf("Input n: ");
```

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

```
    printf("\nInput %d real numbers: ", n);
```

```
    scanf("%f", &x);
```

```
    max = min = x;
```

```
    for (i = 2; i <= n; ++i) {
```

```
        scanf("%f", &x);
```

```
        max = maximum(max, x);
```

```
        min = minimum(min, x);
```

```
    }
```

```
    printf("\n%s%11.3f\n%s%11.3f\n\n",
```

```
        "Maximum value:", max,
```

```
        "Minimum value:", min);
```

```
    return 0;
```

```
}
```

```
float maximum(float x, float y)  
{  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

```
float minimum(float x, float y)  
{  
    if (x < y)  
        return x;  
    else  
        return y;  
}
```

```
void prn_info(void)
```


```
{
```

```
    printf("\n%s\n%s\n\n",
```

```
        "This program reads an integer value  
for n, and then",
```

```
        "processes n real numbers to find max  
and min values.");
```

```
}
```

- 
- Chiama alla funzione `prn_info()`
  - La funzione è una chiamata alla funzione `printf()` per stampare a schermo informazioni
    - Sul programma
    - Sul suo uso

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


```
float  maximum(float x, float y);  
float  minimum(float x, float y);  
void   prn_info(void);
```

```
int main(void)  
{  
    int    i, n;  
    float  max, min, x;  
  
    prn_info();  
    printf("Input n: ");  
    scanf("%d", &n);  
    printf("\nInput %d real numbers: ", n);  
    scanf("%f", &x);  
    max = min = x;  
    for (i = 2; i <= n; ++i) {  
        scanf("%f", &x);  
        max = maximum(max, x);  
        min = minimum(min, x);  
    }  
    printf("\n%s%11.3f\n%s%11.3f\n\n",  
        "Maximum value:", max,  
        "Minimum value:", min);  
    return 0;  
}
```

```
float maximum(float x, float y)  
{  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

```
float minimum(float x, float y)  
{  
    if (x < y)  
        return x;  
    else  
        return y;  
}
```

```
void prn_info(void)  
{  
    printf("\n%s\n%s\n\n",  
        "This program reads an integer value  
for n, and then",  
        "processes n real numbers to find max  
and min values.");  
}
```

- 
- Viene chiesto in input il numero n
  - Viene letto il numero n tramite la scanf()
  - Il numero inserito viene interpretato come un intero decimale tramite la conversione %d



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


```
float  maximum(float x, float y);  
float  minimum(float x, float y);  
void   prn_info(void);
```

```
int main(void)  
{  
    int    i, n;  
    float  max, min, x;  
  
    prn_info();  
    printf("Input n: ");  
    scanf("%d", &n);  
    printf("\nInput %d real numbers: ", n);  
    scanf("%f", &x);  
    max = min = x;  
    for (i = 2; i <= n; ++i) {  
        scanf("%f", &x);  
        max = maximum(max, x);  
        min = minimum(min, x);  
    }  
    printf("\n%s%11.3f\n%s%11.3f\n\n",  
        "Maximum value:", max,  
        "Minimum value:", min);  
    return 0;  
}
```

```
float maximum(float x, float y)  
{  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

```
float minimum(float x, float y)  
{  
    if (x < y)  
        return x;  
    else  
        return y;  
}
```

```
void prn_info(void)  
{  
    printf("\n%s\n%s\n\n",  
        "This program reads an integer value  
for n, and then",  
        "processes n real numbers to find max  
and min values.");  
}
```

- 
- Si chiede all'utente di inserire n numeri reali
  - Si legge il primo e il suo valore è memoria all'indirizzo della variabile x
  - max e min sono inizializzate col valore di x
    - $\text{max} = (\text{min} = x)$
    - $\text{min} = x$  ritorna il valore assegnato

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


```
float  maximum(float x, float y);  
float  minimum(float x, float y);  
void   prn_info(void);
```

```
int main(void)  
{  
    int    i, n;  
    float  max, min, x;  
  
    prn_info();  
    printf("Input n: ");  
    scanf("%d", &n);  
    printf("\nInput %d real numbers: ", n);  
    scanf("%f", &x);  
    max = min = x;  
    for (i = 2; i <= n; ++i) {  
        scanf("%f", &x);  
        max = maximum(max, x);  
        min = minimum(min, x);  
    }  
    printf("\n%s%11.3f\n%s%11.3f\n\n",  
        "Maximum value:", max,  
        "Minimum value:", min);  
    return 0;  
}
```

```
float maximum(float x, float y)  
{  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

```
float minimum(float x, float y)  
{  
    if (x < y)  
        return x;  
    else  
        return y;  
}
```

```
void prn_info(void)  
{  
    printf("\n%s\n%s\n\n",  
        "This program reads an integer value  
for n, and then",  
        "processes n real numbers to find max  
and min values.");  
}
```

- 
- A ogni ciclo si legge un nuovo valore per x
  - Viene stabilito il massimo fra i due valori max e x e si assegna a max

`max = maximum(max, x);`

- Viene stabilito il minimo fra i due valori min e x e si assegna a min

`min = minimum(min, x);`

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


```
float  maximum(float x, float y);  
float  minimum(float x, float y);  
void   prn_info(void);
```

```
int main(void)  
{  
    int    i, n;  
    float  max, min, x;  
  
    prn_info();  
    printf("Input n: ");  
    scanf("%d", &n);  
    printf("\nInput %d real numbers: ", n);  
    scanf("%f", &x);  
    max = min = x;  
    for (i = 2; i <= n; ++i) {  
        scanf("%f", &x);  
        max = maximum(max, x);  
        min = minimum(min, x);  
    }  
    printf("\n%s%11.3f\n%s%11.3f\n\n",  
        "Maximum value:", max,  
        "Minimum value:", min);  
    return 0;  
}
```

```
float maximum(float x, float y)  
{  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

```
float minimum(float x, float y)  
{  
    if (x < y)  
        return x;  
    else  
        return y;  
}
```

```
void prn_info(void)  
{  
    printf("\n%s\n%s\n\n",  
        "This program reads an integer value  
for n, and then",  
        "processes n real numbers to find max  
and min values.");  
}
```

- 
- Definizione della funzione `maximum()`
  - Specifica cosa la funzione farà quando viene chiamata
  - La definizione consiste di una testa e di un corpo
  - La testa è la parte prima della prima {
    - `float maximum(float x, float y)`
    - Il primo float è il tipo del valore ritornato
    - Gli altri due il tipo dei parametri della funzione
    - Entrambe informazioni per il compilatore
  - Il corpo è quello compreso fra le { }

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


```
float  maximum(float x, float y);  
float  minimum(float x, float y);  
void   prn_info(void);
```

```
int main(void)  
{  
    int    i, n;  
    float  max, min, x;  
  
    prn_info();  
    printf("Input n: ");  
    scanf("%d", &n);  
    printf("\nInput %d real numbers: ", n);  
    scanf("%f", &x);  
    max = min = x;  
    for (i = 2; i <= n; ++i) {  
        scanf("%f", &x);  
        max = maximum(max, x);  
        min = minimum(min, x);  
    }  
    printf("\n%s%11.3f\n%s%11.3f\n\n",  
        "Maximum value:", max,  
        "Minimum value:", min);  
    return 0;  
}
```

```
float maximum(float x, float y)  
{  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

```
float minimum(float x, float y)  
{  
    if (x < y)  
        return x;  
    else  
        return y;  
}
```

```
void prn_info(void)  
{  
    printf("\n%s\n%s\n\n",  
        "This program reads an integer value  
for n, and then",  
        "processes n real numbers to find max  
and min values.");  
}
```

- 
- Le due funzioni hanno entrambe parametri x e y.
  - Variabile x usata anche nel main
  - Non c'è nessuna relazione fra i parametri
  - Sono logicamente e fisicamente diversi
  - L'uso dello stesso nome informa che il loro uso è simile.



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


```
float  maximum(float x, float y);  
float  minimum(float x, float y);  
void   prn_info(void);
```


```
int main(void)  
{  
    int    i, n;  
    float  max, min, x;  
  
    prn_info();  
    printf("Input n: ");  
    scanf("%d", &n);  
    printf("\nInput %d real numbers: ", n);  
    scanf("%f", &x);  
    max = min = x;  
    for (i = 2; i <= n; ++i) {  
        scanf("%f", &x);  
        max = maximum(max, x);  
        min = minimum(min, x);  
    }  
    printf("\n%s%11.3f\n%s%11.3f\n\n",  
        "Maximum value:", max,  
        "Minimum value:", min);  
    return 0;  
}
```

```
float maximum(float x, float y)  
{  
    if (x > y)  
        return x;  
    else  
        return y;  
}
```

```
float minimum(float x, float y)  
{  
    if (x < y)  
        return x;  
    else  
        return y;  
}
```

```
void prn_info(void)  
{  
    printf("\n%s\n%s\n\n",  
        "This program reads an integer value  
for n, and then",  
        "processes n real numbers to find max  
and min values.");  
}
```

- 
- L'istruzione `return` restituisce il valore ritornato dalla funzione
  - La forma generale è  
`return expr;`
  - Si può anche avere un  
`return;`  
ma non è considerato molto elegante

- 
- Una maniera diversa di dichiarare le funzioni
  - Va bene per piccoli file

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


```
float maximum(float x, float y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

```
float minimum(float x, float y)
{
    if (x < y)
        return x;
    else
        return y;
}
```

```
void prn_info(void)
{
    printf("\n%s\n%s\n\n",
        "This program reads an integer value
for n, and then",
        "processes n real numbers to find max
and min values.");
}
```

```
int main(void)
{
    int    i, n;
    float  max, min, x;

    prn_info();
    printf("Input n: ");
    scanf("%d", &n);
    printf("\nInput %d real numbers: ", n);
    scanf("%f", &x);
    max = min = x;
    for (i = 2; i <= n; ++i) {
        scanf("%f", &x);
        max = maximum(max, x);
        min = minimum(min, x);
    }
    printf("\n%s%11.3f\n%s%11.3f\n\n",
        "Maximum value:", max,
        "Minimum value:", min);
    return 0;
}
```

- 
- La definizione posta prima dell'uso
  - Il compilatore ha comunque le informazioni necessarie
  - Prototipo dato dalla testa della funzione




# Passaggio per valore

- Esistono due tipi di passaggio di parametri ad una funzione
  - Passaggio per valore
    - Viene creata una copia della variabile passata come parametro
  - Passaggio per riferimento
    - Viene passata la variabile stessa

# Passaggio per valore

- Esistono due tipi di passaggio di parametri ad una funzione
  - Passaggio per valore
    - Viene creata una copia della variabile passata come parametro
  - Passaggio per riferimento
    - Viene passata la variabile stessa
- In C i parametri sono passati solo per valore



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

```
int main(void)
{
    int a = 1;
    void try_to_change_it(int);

    printf("%d\n", a); /* 1 is printed */
    try_to_change_it(a);
    printf("%d\n", a); /* 1 is printed again! */
    return 0;
}
```

```
void try_to_change_it(int a)
{
    a = 777;
}
```