INPUT/OUTPUT

- L'immissione dei dati di un programma e l'uscita dei suoi risultati avvengono attraverso operazioni di lettura e scrittura.

- Il C non ha istruzioni predefinite per l'input/output.

- In ogni versione ANSI C, esiste una *Libreria Standard* (**stdio**) che mette a disposizione alcune funzioni (dette *funzioni di libreria*) per effettuare l'input e l'output da e verso dispositivi.

- **Dispositivi standard di input e di output:**
  - per ogni macchina, sono periferiche predefinite (generalmente tastiera e video).

INPUT/OUTPUT

Le dichiarazioni delle funzioni messe a disposizione da tale libreria devono essere *incluse* nel programma:

`#include <stdio.h>`

- **`#include`** e` una direttiva per il **preprocessore** C:
- nella fase precedente alla compilazione del programma ogni direttiva “#...” viene eseguita, provocando delle modifiche testuali al programma sorgente.
- Nel caso di **`#include <nomefile>:`**

  viene sostituita l’istruzione stessa con il contenuto del file specificato.

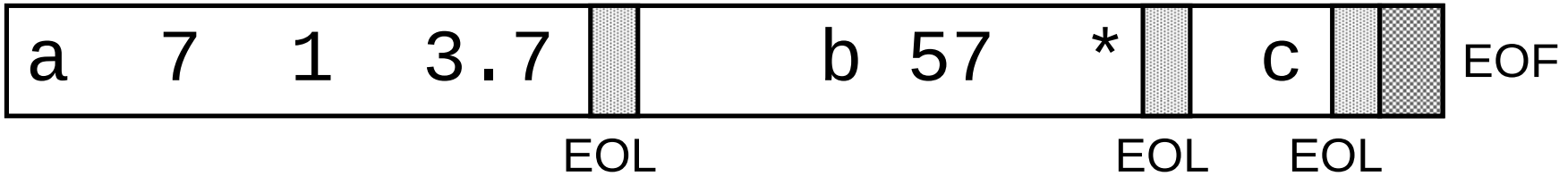
INPUT/OUTPUT

Il C vede le informazioni lette/scritte da/verso i dispositivi standard di I/O come file *sequenziali*, cioè **sequenze di caratteri** (o *stream*).

- Gli *stream* di input/output possono contenere dei caratteri di controllo:
  - End Of File (EOF)
  - End Of Line (EOL)

Sono disponibili funzioni di libreria per:

- Input/Output a caratteri
- Input/Output a stringhe di caratteri
- Input/Output con formato



INPUT/OUTPUT CON FORMATO

- Nell'I/O con formato occorre specificare il formato (*tipo*) dei dati che si vogliono leggere oppure stampare.

- Il formato stabilisce:
  - come interpretare la sequenza dei caratteri immessi dal dispositivo di ingresso (nel caso della lettura)
  - con quale sequenza di caratteri rappresentare in uscita i valori da stampare (nel caso di scrittura)
- Il formato viene specificato mediante apposite ***direttive di formato***; ad esempio **%d**, **%f**, **%s** ecc.

LETTURA CON FORMATO: `scanf`

E’ una particolare forma di assegnamento: la scanf assegna i valori letti alle variabili specificate come argomenti (nell'ordine di lettura).

```
scanf(<stringa-formato>, <sequenza-variabili>);
```

Ad esempio:

```
int X;
float Y;
scanf("%d%f", &X, &Y);
```

LETTURA CON FORMATO: scanf

```
scanf(<stringa-formato>, <sequenza-variabili>);
```

- **scanf** legge una serie di valori in base alle specifiche contenute in **<stringa-formato>** e memorizza i valori letti nelle variabili specificate in **<sequenza-variabili>.**
- Se la **<stringa-formato>** contiene N direttive (del tipo %..), è necessario che le variabili specificate nella **<sequenza-variabili>** siano esattamente N.

- restituisce il numero di valori letti e memorizzati, oppure EOF in caso di end of file :

```
int X, Y, K;
K = scanf("%d%d", &X, &Y);
```

➔ se vengono immessi da input i due valori 100 e -25, le variabili X,Y e K assumeranno i seguenti valori:

X=100 Y=-25 K=2

scanf & formato

Ogni direttiva di formato prevede dei separatori specifici:

Tipo di dato	Direttive di formato	Separatori
Intero	%d, %x, %u, etc.	Spazio, EOL, EOF.
Reale	%f %g etc.	Spazio, EOL, EOF
Carattere	%c	Nessuno
Stringa	%s	Spazio, EOL, EOF

```
int  X; float Y; char Z;
scanf("%d%f%c", &X, &Y, &Z);
```

Osservazioni:

- La <stringa-formato> puo` contenere dei caratteri qualsiasi (che vengono scartati, durante la lettura), che rappresentano separatori aggiuntivi rispetto a quelli standard.
  Ad esempio: **`scanf(“%d:%d:%d”, &A, &B, &C);`**

  richiede che i tre dati da leggere vengano immessi separati dal carattere “**:**”.

SCRITTURA CON FORMATO: `printf`

La **`printf`** viene utilizzata per fornire in uscita il valore di una variabile, o, più in generale, il risultato di una espressione:

```
printf(<stringa-formato>[,<sequenza-elementi>]);
```

- Anche in scrittura è necessario specificare (mediante una **`<stringa-formato>`**) il formato dei dati che si vogliono stampare.
- **`<sequenza-elementi>`** è una lista di ***espressioni*** (tante quante le direttive di formato contenute nella **`<stringa-formato>`).**

Ad esempio:

```
int X=19;
float Y=2.5;
printf("%d%f", X, X+Y);
```

printf

```
printf(<stringa-formato>[,<sequenza-elementi>]);
```

- **printf** scrive una serie di valori in base alle specifiche contenute in **<stringa-formato>.**
- I valori visualizzati sono i risultati delle espressioni indicate nella **<sequenza-elementi>.**
- La **printf** restituisce il numero di caratteri scritti.
- La stringa di formato della **printf** può contenere sequenze costanti di caratteri da stampare (nell'ordine indicato).

Ad esempio:

```
int X=19, K;
float Y=2.5;
K=printf("Risultato: %d%f\n", X, X+Y);
```

Effetti:

```
int X=19, K;
float Y=2.5
```

FORMATI COMUNI

- Formati più comuni:

int	**%d**
float	**%f**
carattere singolo	**%c**
stringa di caratteri	**%s**

- Caratteri di controllo:

newline	**\n**
tab	**\t**
backspace	**\b**
form feed	**\f**
carriage return	**\r**

- Per la stampa del carattere '**%**' si usa: **%%**

ESEMPIO

```
#include <stdio.h>
main()
{int   k;
scanf("%d",&k);
printf("Quadrato di %d: %d\n",k,k*k);
}
```

Esempio:

Se in ingresso viene immesso il dato: **3**

La **printf** stampa:

ESEMPIO

Rivediamo l'esempio visto inizialmente:

```
/*programma che, letti due numeri a terminale, ne stampa
   la somma*/

#include <stdio.h>

main()
{ int X,Y;  /* p. dichiarativa */

  scanf("%d%d",&X,&Y);/*lettura dei due dati*/
  printf("%d",X+Y);/* stampa della loro somma */
}
```

Dati da input i due valori 26 e -32, il programma stampa:

ESEMPIO

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

- Se in ingresso vengono dati:

```
ABC 3 7.345
```

- la **scanf** effettua i seguenti assegnamenti:

```
char c1
char c2
char c3
int i
float x
```

ESEMPIO

```
#include <stdio.h>
main()
{char Nome='A';
char Cognome='C';
printf("%s\n%c. %c. \n%s\n",
         "Programma scritto da:",
               Nome, Cognome,"Fine");
}
```

Stampa:

Esempio

Esempio:

stampa della codifica (decimale, ottale e esadecimale) di un carattere dato da input.

```
#include <stdio.h>

main()
{ char a;

printf("Inserire un carattere:  ");
scanf("%c",&a);
printf("\n%c vale %d in decimale, %o in ottale \
       e %x in hex.\n",a, a, a, a);
}
```

Effetti dell'esecuzione:

```
Inserire un carattere:  A

A vale    in decimale,    in ottale e    in hex.
```

Esercizio

Calcolo dell'orario previsto di arrivo.

Scrivere un programma che legga tre interi positivi da terminale, rappresentanti l'orario di partenza (ore, minuti, secondi) di un vettore aereo, legga un quarto intero positivo rappresentante il tempo di volo in secondi e calcoli quindi l'orario di arrivo.

Prima specifica:

```
main()
{  /*dichiarazione variabili: occorrono tre
   variabili intere per l'orario di partenza ed
   una variabile intera per i secondi di volo.  */
   /*leggi i dati di ingresso */
   /*calcola l'orario di arrivo */
   /*stampa l'orario di arrivo */
}
```

Soluzione:

```
#include <stdio.h>
main()
{ /* dichiarazione dati */

/* leggi i dati di ingresso*/

/* calcola l'orario di arrivo*/

/* stampa l'orario di arrivo*/

;
}
```

Dichiarazioni e Definizioni

Nella parti dichiarative di un programma C possiamo incontrare:

- definizioni (di variabile, o di funzione)
- dichiarazioni (di tipo o di funzione)

Definizione:

Descrive le proprietà dell'oggetto definito e ne determina l'esistenza.

Ad esempio:

```
int V; /* definizione della variabile intera V */
```

Dichiarazione:

Descrive soltanto delle proprietà di oggetti, che verranno (eventualmente) creati mediante definizione.

Ad esempio: *dichiarazione* di un tipo di dato non primitivo:

```
typedef .... newT; /* newT è un tipo non primitivo*/
```

Dichiarazione di tipo

La dichiarazione di tipo serve per introdurre tipi non primitivi.

`typedef <descrizione-nuovo-tipo> newT;`

- si utilizza la parola chiave **`typedef`**.
- la dichiarazione associa ad un tipo di dato non primitivo un identificatore arbitrario (**`newT`**)
- le caratteristiche del nuovo tipo sono indicate in **`<descrizione-nuovo-tipo>`**

L'introduzione di tipi non primitivi aumenta la **leggibilità** e **modificabilità** del programma.

Tipi scalari non primitivi

In C sono possibili dichiarazioni di tipi scalari non primitivi:

- tipi **ridefiniti**
- [tipi **enumerati]** (non li tratteremo)

Tipo ridefinito:

Si ottiene associando un nuovo identificatore a un tipo gia` esistente (primitivo o non).

Sintassi:

`typedef` <id-tipo-esistente> <id-nuovo-tipo> ;

Esempio:

```
typedef int MioIntero; /*  MioIntero è un tipo non
                           primitivo che ridefinsce il
                           tipo int*/
MioIntero X;          /* X è di tipo MioIntero */
int Y;                /* Y è di tipo int */
```

➔ X e Y rappresentano entrambi valori interi, ma ***nominalmente*** sono di tipo diverso.

Equivalenza tra tipi di dato

Quando due variabili hanno lo stesso tipo?
Dipende dalla ***realizzazione*** del linguaggio.

In generale, vi sono due possibilita` :

- equivalenza **strutturale**
- equivalenza **nominale**

Equivalenza strutturale:

due dati sono considerati di tipo equivalente se hanno la stessa struttura.

Ad esempio:

```
typedef int MioIntero;
MioIntero X;
int Y;
```

Se la realizzazione di C prevede equivalenza strutturale:

X e Y sono **dello stesso tipo**.

Equivalenza tra tipi di dato

Equivalenza nominale:

due dati sono considerati di tipo equivalente se sono stati definiti usando lo stesso identificatore di tipo.

Ad esempio:

```
typedef int MioIntero;
MioIntero X;
int  Y;
```

Se la realizzazione di C prevede equivalenza nominale:

X e Y sono **di tipo diverso**.

Equivalenza tra tipi di dato

L'equivalenza nominale è più restrittiva:

non è detto che dati strutturalmente equivalenti siano anche nominalmente equivalenti (dipende dalla realizzazione del linguaggio!)

Equivalenza di tipo in C:

- Lo standard non stabilisce il tipo di equivalenza da adottare.
- Per garantire la portabilità, è necessario sviluppare programmi che presuppongano una realizzazione basata su equivalenza nominale.