I/O in linguaggio C

I/O di caratteri e stringhe

- Tutti i caratteri stampabili sono rappresentabili dal tipo `char`.
- Le costanti di tipo `char` possono essere specificate da un valore numerico o nel formato `'`⟨carattere⟩`'`.

```
char c1='A', c2=96; /* c1==c2==<la lettera A>
```

- esistono costanti carattere particolari.

I/O di caratteri e stringhe

Escape Sequence	Name	Meaning
`\a`	Alert	Produces an audible or visible alert
`\b`	Backspace	Moves the cursor back one position (non-destructive)
`\f`	Form Feed	Moves the cursor to the first position of the next page
`\n`	New Line	Moves the cursor to the first position of the next line
`\r`	Carriage Return	Moves the cursor to the first position of the current line
`\t`	Horizontal Tab	Moves the cursor to the next horizontal tabular position
`\v`	Vertical Tab	Moves the cursor to the next vertical tabular position
`\'`		Produces a single quote
`\"`		Produces a double quote
`\\`		Produces a single backslash
`\0`		Produces a null character

I/O di caratteri e stringhe

```
int getchar(void);
```

- Legge da `stdin` un carattere;
- ritorna il carattere letto se non si verificano errori in lettura.
- Se si verificano errori o se si incontra un segnale di fine-file ritorna la costante `EOF` (**E**nd-**O**f-**F**ile).

Nota: `getchar()` ritorna un `int` per poter rappresentare `EOF` con un valore che non sia un `char`.

```
int putchar(int ch);
```

- Emette il carattere `ch` su `stdout`;
- ritorna `ch` se non ci sono errori di output, altrimenti `EOF`.

I/O di caratteri e stringhe

```
char *gets(char *s);
char *gets(char s[]);
```

- Il parametro `s` è una stringa.
- Legge i successivi caratteri da `stdin`, fino a quando incontra un `'\n'`.
- I caratteri precedenti `'\n'` vengono copiati in `s`, seguiti da `'\0'` (`'\n'` non viene memorizzato).
- Ritorna `s` (puntatore), oppure `NULL` se si verificano errori in input.

I/O di caratteri e stringhe

```
int puts(char *s);
int puts(char s[]);
```

- Stampa la stringa `s` — tranne il terminatore `'\0'` — su `stdout`.
- Al termine della stampa emette anche un `'\n'` per ritornare a capo.
- Ritorna `EOF` se si verificano errori di output,
- altrimenti un valore ≥ 0.

Output formattato

```
int printf(char *format, arg1, arg2, ...);
```

- Funzione con numero variabile di parametri.
- `format` è una *stringa-formato.*
- `printf` converte gli argomenti `arg1`, `arg2`, `...` in stringhe, e li emette su `stdout` secondo le indicazioni contenute in `format`.
- Gli elementi della stringa `format` sono di due tipi
  - caratteri normali, copiati su `stdout`;
  - specifiche di conversione, che specificano come visualizzare `arg1`, `arg2`, ...
- Le specifiche di conversione sono sequenze speciali che hanno % come primo carattere.

Output formattato

Esempio. Il frammento di codice

```
int x[]={4,5,9,2,1};
...
for(i=0; i<5; i++)
  printf( "x[%d]=%d\n", i, x[i]);
```

produce l'output

```
x[0]=4
x[1]=5
x[2]=9
x[3]=2
x[4]=1
```

- %d è la specifica associata ad un intero;
- verrà sostituita in output con la rappresentazione dell'argomento corrispondente.

Output formattato

Una specifica di conversione ha il formato

$$\%\langle \text{width} \rangle_{\text{opt}} \langle \texttt{.prec} \rangle_{\text{opt}} \langle \text{spec} \rangle$$

dove

- ⟨width⟩ è un numero opzionale che specifica l'ampiezza minima — in caratteri — con cui deve essere stampato l'argomento.
- ⟨`.prec`⟩ è un'informazione opzionale per argomenti numerici, che indica che l'argomento corrispondente va stampato con *prec* cifre dopo la virgola.
- ⟨spec⟩ è uno tra i caratteri `dioxXucsfeEgGp%` e determina il tipo di argomento e la sua rappresentazione in output.

Output formattato

- `printf` mette in corrispondenza le specifiche di conversione *nell'ordine in cui queste appaiono* in `format` con i successivi argomenti.
- È compito del chiamante (praticamente: del programmatore) fare in modo che ad ogni specifica venga associato un argomento del tipo corretto.

```
double x=3.0;
int y=7;
...
printf( "x=%d, y=%d\n", x, y); /* Errore */
```

La `printf` qui ha un comportamento *indefinito:* %d è la specifica per variabili intere, mentre l'argomento corrispondente è un `double`.

Output formattato

Principali tipi di specifiche.

⟨spec⟩	Tipo `arg`	Formato output
`d`, `i`	`int`	Numero decimale
`u`	`unsigned int`	Numero decimale senza segno
`c`	`char`	Carattere singolo
`s`	`char *`	Stringa
`f`	`double`	Reale in formato $m.dddddd$
`e`, `E`	`double`	Reale in formato scientifico $m.ddddddE \pm XX$
`g`, `G`	`double`	Reale nel formato più compatto tra %e e %f
%	—	Stampa un normale %

Output formattato

Esercizio. Scrivere un programma che genera e stampa una tabella di conversione gradi-Celsius → gradi Fahreneit.

```
+---------+---------+
|Celsius  |  Fahr.  |
+---------+---------+
|       0 |   32.00 |
|       1 |   33.80 |
|       2 |   35.60 |
|       3 |   37.40 |
|       4 |   39.20 |
|       5 |   41.00 |
|       6 |   42.80 |
|       7 |   44.60 |
|       8 |   46.40 |
|       9 |   48.20 |
|      10 |   50.00 |
+---------+---------+
```

Output formattato

Formula di conversione:

$$T_F = \frac{9}{5} T_C + 32.$$

Realizzare le seguenti funzioni.

```
int genera_tabella(int from, int to, int step,
                   int *c, double *res);
void stampa_tabella(int *cel,
                    double *fahr, int len);
```


Output formattato

```
int genera_tabella(int from, int to, int step,
                   int *c, double *res);
```

Riceve i parametri interi `from`, `to`, `step` e i vettori `c` e `res`.
Effettua la conversione $T_C \to T_F$ per ogni valore intero

`from` $\leq T_C \leq$ `to` con passo `step`.

Memorizza i valori di T_C e T_F così calcolati in `c[]` e `res[]`.
Cioè, in uscita: `c[i]` $=$ `from` $+$ `i` $\cdot$ `step`;
`res[i]`=la temp. in ^oF corrispondente a `c[i]`.
Ritorna il numero di valori generati.
Notare che `c` e `res` devono essere vettori lunghi a sufficienza (a cura del chiamante!)

Output formattato

```
void stampa_tabella(int *cel,
                    double *fahr, int len);
```

- Riceve i due vettori `cel` e `fahr`, e la loro lunghezza `len`.
- Stampa l'intestazione della tabella

```
+--------+--------+
|Celsius |  Fahr. |
+--------+--------+
```

- Stampa le coppie di valori

```
| cel[i] | fahr[i]|
```

  per `i=0...len-1`.

- I valori `fahr[i]` devono avere 2 cifre decimali ed essere stampati su campi di 6 caratteri.
- I valori `cel[i]` devono essere stampati su campi di 6 caratteri.

Output formattato

```
int genera_tabella(int from, int to, int step,
                   int *c, double *res)
{
  int i;

  for(i=0;from<=to; from+=step, i++) {
    c[i]=from;
    res[i]=from*(9.0/5.0) +32.0;
  }

  return i;
}
```

Output formattato

```
void stampa_tabella(int *cel, double *fahr,
                    int len)
{
  int i;

  printf( "+--------+--------+\n" );
  printf( "|Celsius |  Fahr. |\n" );
  printf( "+--------+--------+\n" );
  for(i=0; i<len; i++) {
    printf( "| %6d | %6.2f |\n",
            cel[i], fahr[i] );
  }
  printf( "+--------+--------+\n" );
}
```


Output formattato

```
#define MAXLEN 100
int main(void)
{
  int cel[MAXLEN];
  double fahr[MAXLEN];
  int num;

  num=genera_tabella(0,10,1,cel,fahr);
  stampa_tabella(cel, fahr, num);
}
```

Input formattato

```
int scanf(char *format, arg1, arg2, ...);
```

- È “duale” alla `printf`.
- `format` contiene
  - caratteri di spaziatura (ignorati: `' '`, `'\n'`, `'\t'`,...);
  - caratteri normali diversi da % che devono coincidere con i successivi caratteri non-spazio in input.
  - specifiche di conversione, analoghe a quelle di `printf`.
- Ritorna il numero di argomenti che è riuscita a leggere correttamente

Input formattato

Principali tipi di specifiche.

⟨spec⟩	Tipo `arg`	Formato input
`d`	`int *`	Numero decimale
`c`	`char *`	Carattere singolo (anche spazio)
`s`	`char *`	Stringa *senza spazi*
`e`, `f`, `g`	`float *`	Numero reale, in qualunque formato
%	—	Il prossimo carattere in input deve essere un %

Input formattato

Per leggere un argomento, `scanf` fa quanto segue.

- Salta i successivi spazi (se la specifica non è %c);
- cerca una sequenza massimale di caratteri che può rappresentare l'argomento del tipo specificato (intero, stringa, `...`).
- se la trova, scrive il valore opportuno nell'argomento,
- altrimenti ritorna senza leggere l'argomento.
- I caratteri non-spazio in `format` tra un argomento e l'altro devono essere riscontrati nell'ordine in cui appaiono.
- Come per `printf`, bisogna garantire la corretta corrispondenza tra specifiche di conversione e tipi degli argomenti.

Input formattato

Esempio.

```
char str[100];
int x, num;
...
num=scanf("%s=%d", str, &x); // str e‘ gia‘ punta
```

Ritorna `num=2` (tutto OK) se si digita

`␣␣pippo␣=␣3`

ma ritorna `num=1` se si digita

`␣␣pippo␣=␣abc`

oppure

`␣␣pippo␣;␣3`

Negli ultimi due casi `x` rimane indefinito.

Buffer di I/O

Alcune note.

- I dispositivi di I/O `stdin` e `stdout` sono gestiti a tutti gli effetti come file, e dispongono di un *buffer* nel quale transitano i caratteri letti/scritti.
- Questo può provocare a volte effetti corretti ma inaspettati nelle operazioni di I/O.
- Conviene rappresentarsi `stdin` come una lunga "stringa" (=il buffer) che viene "consumata" carattere per carattere dalle funzioni di input.
- Un carattere non letto (non consumato) rimane nel buffer, e viene sottoposto alla prossima operazione di input.

Buffer di I/O

Esempio. Il seguente codice dovrebbe chiedere un numero all'utente, poi fare una pausa e aspettare una conferma.

```
printf("Inserisci un numero:");
scanf( "%d", &x );
printf( "Continua?");
ch=getchar();
if( ch=='y' ) ...
```

- Se l'utente digita la sequenza

1	2	3	\n	

non c'è alcuna pausa. Perché?

- La `scanf` consuma correttamente il numero 123, poi *non consuma* il `'\n'`,
- che verrà letto immediatamente dopo da `getchar()`.
- Quindi si avrà `ch=='\n'`, subito dopo la richiesta `Continua?`.

Buffer di I/O

- Imprevisti analoghi possono verificarsi in output, sempre per la presenza di un buffer.
- Alcune stampe possono non apparire immediatamente (la sequenzialità delle stampe è comunque sempre garantita!)
- Particolarmente fastidioso durante il *debugging.*
- Per svuotare il buffer di un dispositivo e riportarlo in uno stato iniziale vuoto, è disponibile la funzione `fflush`.

Buffer di I/O

Rimedi.

- Per l'output: l'istruzione

  ```
  fflush(stdout)
  ```

  svuota il buffer di output, causando la stampa immediata di tutti i caratteri in esso contenuti.
- Da usare quando è necessario "sincronizzare" le operazioni di output.
- Attenzione: la corrispondente

  ```
  fflush(stdin)
  ```

  non garantisce lo svuotamento del buffer di input (lo standard definisce il comportamento di `fflush` solo su buffer di output).

Buffer di I/O

- Nel caso della pausa nel codice precedente conviene cambiare:

```
scanf("%d", &x); fflush(stdin);
printf( "Continua?" );
do {
  ch=getchar();
} while( ch!='y' && ch!='n');
if( ch=='y' ) ...
```

- Se si digita

1	2	3	\n	a	b	c

- `scanf` legge correttamente 123; i caratteri successivi (\nabc) vengono eliminati dal ciclo di lettura;
- la `getchar()` si pone in attesa a buffer vuoto.

I/O su file

Stream

- Un file è un archivio di informazioni organizzato su memoria di massa, associato ad un “nome”.
- Uno stream è una sorgente o destinazione di dati (un file, oppure un dispositivo di I/O come tastiera, schermo, `...`).
- La libreria standard del C supporta stream di testo e binari.
- Uno stream di testo è una sequenza di righe; ogni riga ha uno o più caratteri ed è terminata da ‘`\n`’.
- Uno stream binario è una sequenza “grezza” di byte.
- Uno stream è associato ad un valore di tipo `FILE *` (puntatore a file).
- Il tipo `FILE` è (di solito) un tipo `struct` i cui dettagli sono dipendenti dall’implementazione.

Stream

- Le funzioni che operano su stream sono definite in `stdio.h` (usare `#include <stdio.h>`).
- Ci sono sempre tre stream attivi:
  - `stdout`, per operazioni di scrittura su terminale;
  - `stdin`, per operazioni di lettura da terminale;
  - `stderr`, per messaggi di errore.

Stream

Operazioni sui file.

- Apertura;
- lettura;
- scrittura;
- chiusura;
- cancellazione;
- cambio di nome.

Stream

```
FILE *fopen(char *filename, char *mode)
```

- Apre il file di nome `filename`;
- lo associa ad uno stream (valore `FILE *`);
- restituisce lo stream, oppure `NULL` se l'operazione è fallita.
- `mode` è una stringa che può essere:
  - `"r"`, per aprire il file in lettura;
  - `"w"`, per aprire/creare in scrittura (cancella i contenuti vecchi);
  - `"a"`, append: accoda ai vecchi contenuti;
  - `"r+`, aggiornamento (lettura e scrittura);
  - `"w+"`, aggiornamento (scartando i vecchi contenuti).

Stream

Esempio.

```
FILE *fp;
char fname[64];

printf( "Nome del file: ");
scanf("%s", fname);
fp=fopen(fname, "r");
if( fp==NULL ) {
  printf( "Impossibile aprire %s in lettura\n",
          fname);
}
else {...}
```


Stream

```
int fclose(FILE *fp)
```

- Chiude il file associato allo stream `fp`;
- ritorna 0 se l'operazione ha successo,
- ritorna `EOF` se si è verificato un errore;

```
FILE *fp=fopen("pippo.dat", "w");
...  /* Scrittura dati... */

if( fclose(fp)==EOF ) /* Poco comune. */
  printf("Errore in chiusura file\n");
```

- L'uso più comune è semplicemente

```
fclose(fp);  /* Senza controlli.  */
```

Stream

```
int fflush(FILE *stream)
```

- Le operazioni di I/O (di solito) non sono dirette, ma usano un *buffer* in memoria che viene sincronizzato periodicamente con il contenuto su periferica (disco, schermo, `...`).
- Se `stream` è aperto in scrittura, sincronizza immediatamente i contenuti con il buffer di uscita.
- Ritorna 0 se l'operazione riesce, `EOF` altrimenti.
- Chiamando `fclose(stream)` si causa anche una `fflush(stream)` implicita.
- L'effetto di `fflush(stream)` è indefinito se `stream` è aperto in lettura.

Stream

```
int remove(char *fname)
```

- Cancella il file di nome `fname`; ritorna 0 se l'operazione ha successo, altrimenti un valore diverso da 0.

```
int rename(char *fname, char *newname)
```

- Cambia il nome del file chiamato `fname` in `newname`; ritorna 0 se l'operazione ha successo, altrimenti un valore diverso da 0.

Stream di testo

I/O di caratteri.

```
int fgetc(FILE *stream)
```

- Ritorna il successivo carattere contenuto in `stream`, oppure `EOF` se incontra la fine del file.
- `fgetc(stdin)` equivale a `getchar()`.

```
int fputc(int c, FILE *stream)
```

- Scrive il carattere `c` su `stream` (o meglio, nel buffer di output associato a `stream`).
- `fputc(c, stdout)` equivale a `putchar(c)`.
- Ritorna `c`, oppure `EOF` in caso di errore.

Stream di testo

```
int ungetc(int c, FILE *stream)
```

- rimanda il carattere `c` (convertendo da `int` a `char`) a `stream`.
- `stream` deve essere già aperto in lettura.
- Il carattere sarà letto dalla prima operazione di input successiva.
- `c` non può essere `EOF`.
- Si garantisce la ricollocazione di un solo carattere.

Stream di testo

I/O di stringhe.

```
char *fgets(char *s, int n, FILE *stream)
```

- Legge al più `n-1` successivi caratteri da `stream`,
- oppure fino al primo '`\n`' che incontra in `stream`,
- memorizzandoli nell'array (puntato da) `s`, e scrivendo il terminatore '`\0`' alla fine;
- l'eventuale '`\n`' viene scritto in `s`.
- Ritorna `s` se l'operazione va a buon fine, altrimenti `NULL`.

Stream di testo

Nota. Un frammento come

```
char buf[128];
if( fgets(buf, 128, stdin)!= NULL ) { ... }
```

è analogo a

```
char buf[128];
if( gets(buf)!= NULL ) { ... }
```

ma è più sicuro (non può uscire dall'array).

Stream di testo

```
int fputs(char *s, FILE *stream)
```

- Scrive la stringa `s` su `stream`;
- non aggiunge '`\n`', per andare a capo bisogna fornirlo esplicitamente in `s`.
- Ritorna un valore >=0 se l'operazione è riuscita, altrimenti `EOF`.

Stream di testo

```
int feof(FILE *stream)
```

- Ritorna un numero diverso da 0 se e solo se si è giunti alla fine di `stream`.

Stream di testo

I/O formattato.

```
int fprintf( FILE *stream, char *format,
             arg1, arg2, ...)
```

- Scrive in formato testo i valori di `arg1`, `arg2`, ecc. che seguono `format` su `stream` (aperto in scrittura);
- il significato di `format` è identico alla stringa formato di `printf()`, con le stesse prescrizioni sui tipi degli argomenti.
- Ritorna il numero di caratteri scritti, oppure un valore <0 in caso di errore.

Stream di testo

```
int fscanf( FILE *stream, char *format,
            arg1, arg2, ...)
```

- Legge da `stream` valori di tipo opportuno (secondo le specifiche di `format`) e le pone in (variabili puntate da) `arg1`, `arg2`, ecc.
- `stream` deve essere aperto in lettura.
- il significato di `format` è identico alla stringa formato di `printf()`, con le stesse prescrizioni sui tipi degli argomenti.
- Ritorna il numero di argomenti letti con successo.

Stream di testo

Esercizio. Scrivere un programma che legge da tastiera il nome di un file di testo di input, il nome di un file di uscita, e che scrive nel file di uscita le stesse righe ma *numerate* del file di ingresso.

Esempio. Se il file di ingresso contiene

```
Dear Ms. Andersen,
We submitted a paper about one year ago,
but got no answer from you since then.
```

il file di uscita conterrà

```
1. Dear Ms. Andersen,
2. We submitted a paper about one year ago,
3. but got no answer from you since then.
```


Stream di testo

```
#include <stdio.h>

int main()
{
  FILE *fp1, *fp2;
  char fname1[64], fname2[64];
  char buf[128];
  int count=0;

  printf( "File di input:" );
  scanf( "%s", fname1);
  printf( "File di output:" );
  scanf( "%s", fname2);
  fp1=fopen(fname1,"r");
  if( fp1==NULL ) {
    fprintf( stderr, "Errore in apertura di %s\n", fname1 );
    return 1;
  }
```

Stream di testo

```
  fp2=fopen(fname2,"w");
  if( fp2==NULL ) {
    fprintf( stderr, "Error in apertura %s\n", fname2 );
    return 1;
  }

  while( !feof(fp1) ) {
    if( fgets(buf,128,fp1)!=NULL )
      fprintf( fp2, "%d. %s\n", ++count, buf );
  }

  fclose(fp1);
  fclose(fp2);
}
```