Talk-About
Uso di stdio su Arduino

Il caso di printf

Gruppo Utenti Linux Livorno

https://linux.livorno.it

Obiettivo

Non usare i metodi dell'oggetto `Serial` perché è specifico di Arduino e al suo posto usare la funzione `printf` che è una funzione standard del linguaggio C e del linguaggio C++. Ciò permette di riutilizzare più facilmente lo stesso codice sia su Arduino, sia su un PC dove è più semplice fare il debug.

Su Wikipedia: linguaggio C, linguaggio C++, debug.

stdio

stdio è una libreria che fa parte della libreria standard del C e contiene funzioni per lavorare con i file. Nella filosofia UNIX ogni cosa è un file, non è per forza qualcosa che si trova su un disco, può essere anche un collegamento seriale.

Nell'ambiente di Arduino la stdio è fornita da avr-libc, nell'ambiente Linux è fornita da glibc.

Negli ambienti più comuni, prima di usare le funzioni è necessario includere esplicitamente il file *header* `stdio.h`:

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

Ciò non è necessario nell'ambiente Wiring (e quindi nell'IDE di Arduino) perché quel file è incluso automaticamente.

Questo file *header* contiene fra le altre cose la definizione del puntatore a file che è l'indirizzo di una struttura che contiene varie informazioni sul file; è una struttura di dati cosiddetta *opaca*, cioè a cui il programmatore non deve accedere direttamente, ma deve usare le funzioni fornite dalla libreria. Esistono 3 file convenzionali, che sono già aperti quando il programma è in esecuzione, ai quali ci si riferisce con le seguenti variabili predefinite:

0. `stdin` lo standard input, generalmente la tastiera
1. `stdout` lo standard output, generalmente un terminale
2. `stderr` lo standard error, generalmente un terminale

I numeri non sono messi a caso ;-) corrispondono alle macro `STDIN_FILENO`, `STDOUT_FILENO` e `STDERR_FILENO` in `unistd.h` e sono anche i numeri dei descrittori di file disponibili in bash come `/dev/fd/NUMERO` e tramite la redirezione con `<`, `>` e `>>`.

Su Wikipedia: libreria standard del C, stdio.h

Vedere anche: `man stdio`, `man stdin`, `man bash` e cercare REDIRECTION

printf

Il *prototipo* della funzione `printf` è:

```
int printf(const char *format, ...);
```

Il primo argomento, `format`, è obbligatorio ed è una stringa che contiene testo normale e comandi di formattazione, qui ricordo solo:

- %d e %ld per int e long int (interi e interi lunghi con segno)
- %u e %lu per unsigned int e unsigned long int (interi e interi lunghi senza segno)
- %s per le stringhe terminate da NUL, cioè da `'\0'`
- %c per i caratteri singoli

I tre punti … indicano che la funzione si aspetta un numero facoltativo e variabile di argomenti; essi devono essere coerenti con i comandi di formattazione.

Vedere anche: `man 3 printf`

Torniamo al nostro obiettivo

Vedremo che è possibile usare `printf` al posto di `Serial.print`, ma non è possibile eliminare del tutto l'uso dell'oggetto `Serial` perché è il modo per usare la seriale che funziona su tutte le versioni di Arduino.

Esempio: stampare una data con barre di separazione

`int giorno=13, mese=9, anno=2019;` (sarebbe meglio usare unsigned int)

con `Serial.print` devo scrivere:

```
Serial.print(giorno);
Serial.print('/');
Serial.print(mese);
Serial.print('/');
Serial.println(anno);
```

con `printf` devo scrivere:

```
printf("%d/%d/%d\n", giorno, mese, anno);
```

Se voglio che i numeri dei da 1 a 9 siano preceduti da uno zero nel caso di Serial.print devo aggiungere un controllo esplicito:

```
if (giorno < 9) {
    Serial.print('0');
}
// …
if (giorno < 9) {
```

```
    Serial.print('0');
}
```

con `printf` uso l'opzione per inserire automaticamente uno zero:

```
printf("%02d/%02d/%d\n", giorno, mese, anno);
```

In altri casi fa comodo allineare a destra o a sinistra:

1. `printf("%02d/%04d/%08d<\n", 1234, 1234, 1234);`
2. `printf("%0-2d/%0-4d/%0-8d<\n", 1234, 1234, 1234);`
3. `printf("%s/%s/%s<\n", "a", "a", "a");`
4. `printf("%2s/%4s/%8s<\n", "a", "a", "a");`
5. `printf("%-2s/%-4s/%-8s<\n", "a", "a", "a");`

stampano rispettivamente:

1. `1234/1234/00001234<`
2. `1234/1234/1234    <`
3. `a/a/a<`
4. `` a/   a/       a< ``
5. `a /a   /a       <`

Nel caso dei numeri, la cifra indica il numero minimo di cifre da stampare, ma se il numero è grande viene stampato comunque tutto (ho aggiunto il segno di minore solo per far vedere dove finiscono le stringhe che hanno gli spazi a destra).

Vedere anche: `man 3 printf`, avr-libc

avr-libc

La libreria avr-libc implementa più versioni della funzione `printf` ma Arduino non permette di scegliere quale usare. In ogni caso, siccome la libreria avr-libc è generica, è necessario implementare una o entrambe le funzioni che gestiscono la scrittura e la lettura sulla nostra piattaforma hardware (Arduino!) e richiamare l'apposita funzione che imposta la configurazione:

```
int serial_put(char c, FILE *stream) { /* … */ }
int serial_get(FILE *stream) { /* … */ }
/* … */
FILE stream;
```

```
fdev_setup_stream(&stream, serial_put, serial_get,
_FDEV_SETUP_RW);
```

Vedere anche: avr-libc

Appendice

Sorgente

```
/**
 * Questo esempio mostra come usare printf() e getchar() usando i meccanismi
 * forniti dalla avr-libc. Dopo quella inizializzazione si possono usare
 * altre macro e funzioni definite in stdio.h
 * http://avr-libc.nongnu.org/user-manual/group__avr__stdio.html
 *
 * Daniele Forsi 16/09/2019, CC0
 */

#define SERIAL_BAUD_RATE 9600

void setup_serial(int baudrate) {
  Serial.begin(baudrate);
}

int serial_put(char c, FILE *stream) {
  Stream *object = fdev_get_udata(stream);

  return object->print(c);
}

int serial_get(FILE *stream) {
  Stream *object = fdev_get_udata(stream);

  return object->read();
}

void setup_stdio(Stream& object) {
  static FILE stream;

  fdev_setup_stream(&stream, serial_put, serial_get, _FDEV_SETUP_RW);
  fdev_set_udata(&stream, &object);

  //stdin = stdout = stderr = &stream;
}
```

```
/**
 * Questo demo usa solo stdout perché mostra solo l'uso di printf() però per completezza
 * ho messo anche stdin, stderr, serial_get e passo l'oggetto Serial come parametro
 */
void demo(void) {
  int giorno=13, mese=9, anno=2019;

  // stampa: giorno, mese, anno: giorno 13, mese 9, anno 2019
  printf("giorno, mese, anno: giorno %d, mese %d, anno %d\n", giorno, mese, anno);

  Serial.print("giorno, mese, anno: giorno ");
  Serial.print(giorno);
  Serial.print(", mese ");
  Serial.print(mese);
  Serial.print(", anno ");
  Serial.println(anno);

  printf("\n");

  // stampa: 13/9/2019
  printf("%d/%d/%d\n", giorno, mese, anno);

  Serial.print(giorno);
  Serial.print('/');
  Serial.print(mese);
  Serial.print('/');
  Serial.println(anno);

  printf("\n");

  long int numero=123456789;

  // stampa: numero decimale, 123456789, esadecimale 0x75bcd15
  printf("numero decimale %ld, esadecimale 0x%lX\n", numero, numero);

  Serial.print("numero decimale ");
  Serial.print(numero);
  Serial.print(", esadecimale 0x");
  Serial.println(numero, HEX);

  printf("\n");
```

```
  // stampa: numero negativo decimale -123456789, esadecimale 0xF8A432EB
  printf("numero negativo decimale %ld, esadecimale 0x%lX\n", -numero, -numero);

  Serial.print("numero negativo decimale ");
  Serial.print(-numero);
  Serial.print(", esadecimale 0x");
  Serial.println(-numero, HEX);

  printf("\n*** fine del demo ***\n");
}

void setup(void) {
  setup_serial(SERIAL_BAUD_RATE);
  setup_stdio(Serial);

  demo();
}

void loop(void) { }
```