

Standard I/O

Franco Maria Nardini

Standard I/O Library

- Definita da ISO C
 - implementata non solo in sistemi operativi UNIX
- Gestisce
 - l'allocazione del buffer di I/O
 - l'I/O in blocchi ottimizzati senza bisogno di preoccuparsi di ciò
- Scritta da Ritchie nel 1975
 - pochissime modifiche da quel design iniziale in 40 anni

Idea alla base

- L'I/O visto finora è centrato sul concetto di *file descriptor*
 - dall'apertura di un file si ha un file descriptor che viene usato per tutte le successive operazioni
- La libreria di standard I/O è:
 - centrata sul concetto di *stream* (?)
 - quando si apre o crea un file, stiamo associando uno *stream* al file

Idea alla base

- Gestisce single-byte (ASCII) e multibyte (?) character sets
- no orientamento dello stream alla creazione
- l'uso di funzioni di I/O multibyte su stream non orientato settano l'orientamento

Buffering

- la libreria di standard I/O introduce buffering
 - si vuol usare il minimo numero di chiamate a `read` e `write`
 - differenziato per stream

Tre tipi di buffering

- fully buffered: l'I/O è effettivamente fatto quando lo standard I/O buffer è pieno.
 - I/O su disco è generalmente fully buffered.
 - Il buffer è creato con una malloc la prima volta che lo stream è usato
 - *flush* è il termine con cui si indica lo svuotamento del buffer:
 - automatico, quando il buffer è pieno
 - manuale, tramite `fflush()`

Tre tipi di buffering

- line buffered: l'I/O è effettivamente fatto quando si incontra un '\n'.
 - ciò consente di fare I/O di caratteri (fputc()) sapendo che il vero I/O ci sarà alla scrittura di un '\n'.
- Attenzione!
 - dimensione del buffer è fissata: I/O può avvenire prima del '\n' in caso di linee lunghe.
 - nel momento in cui si richiede input da uno stream line buffered:
 - tutti gli stream line buffered in output sono flushati

Tre tipi di buffering

- unbuffered: nessun buffer associato.
 - 10 caratteri su uno stream unbuffered escono *il prima possibile*
- STDERR è generalmente unbuffered (?)

Buffering

- ISO C richiede
 - STDIN e STDOUT sono fully buffered
 - se non riferiscono a device interattivi (terminale)
 - STDERR mai fully buffered
- Le principali implementazioni:
 - STDIN e STDOUT sono line buffered se riferiscono a device interattivo, altrimenti fully buffered
 - STDERR sempre unbuffered

Flush e chiusura

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

```
int fclose(FILE fp);
```

Returns: 0 if OK, EOF on error

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

```
int fflush(FILE fp);
```

Returns: 0 if OK, EOF on error

Letture e scrittura

- Si può interagire con tre tipi di I/O non formattato:
 - I/O di un carattere alla volta
 - I/O di una riga alla volta
 - I/O *diretto*

Lettura per carattere

```
#include <stdio.h>

int getc(FILE fp);

int fgetc(FILE fp);

int getchar(void);
```

All three return: next character if OK, EOF on end of file or error

- ritorno di un unsigned char come int
- int necessario per EOF o errori (negativi, -1 spesso)
- non si confronta il ritorno char con EOF
- stessi valori per errore o EOF

Come distinguere?

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

```
int ferror(FILE fp);
```

```
int feof(FILE fp);
```

Both return: nonzero (true) if condition is true, 0 (false) otherwise

```
void clearerr(FILE fp);
```

Scrittura per carattere

```
#include <stdio.h>

int putc(int c, FILE fp);

int fputc(int c, FILE fp);

int putchar(int c);
```

All three return: *c* if OK, EOF on error

Lettura per riga

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

```
char fgets(char restrict buf, int n, FILE restrict fp);
```

```
char gets(char buf);
```

Both return: `buf` if OK, `NULL` on end of file or error

- differenze (?)
- non usare `gets()`: buffer overflow

Scrittura per riga

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

```
int fputs(const char *restrict str, FILE *restrict fp);
```

```
int puts(const char str);
```

Both return: non-negative value if OK, EOF on error

- `puts ()` scrive su STDOUT (con newline)
- `fputs ()` richiede la gestione del newline

Performance

Function	User CPU (seconds)	System CPU (seconds)	Clock time (seconds)	Bytes of program text
<code>fgets, fputs</code>	2.27	0.30	3.49	143
<code>getc, putc</code>	8.45	0.29	10.33	114
<code>fgetc, fputc</code>	8.16	0.40	10.18	114

Output Formattato

```
#include <stdio.h>
int printf(const char *restrict format, ...);
int fprintf(FILE *restrict fp, const char *restrict format, ...);
int dprintf(int fd, const char *restrict format, ...);
```

All three return: number of characters output if OK, negative value if output
error

Conversioni

Conversion type	Description
d, i	signed decimal
o	unsigned octal
u	unsigned decimal
x, X	unsigned hexadecimal
f, F	double floating-point number
e, E	double floating-point number in exponential format
g, G	interpreted as f, F, e, or E, depending on value converted
a, A	double floating-point number in hexadecimal exponential format
c	character (with l length modifier, wide character)
s	string (with l length modifier, wide character string)
p	pointer to a void
n	pointer to a signed integer into which is written the number of characters written so far
%	a % character
C	wide character (XSI option, equivalent to lc)
S	wide character string (XSI option, equivalent to ls)

Input Formattato

```
#include <stdio.h>
int scanf(const char *restrict format, ...);
int fscanf(FILE *restrict fp, const char *restrict format, ...);
int sscanf(const char *restrict buf, const char *restrict format, ...);
```

All three return: number of input items assigned,
EOF if input error or end of file before any conversion

Conversioni

Conversion type	Description
d	signed decimal, base 10
i	signed decimal, base determined by format of input
o	unsigned octal (input optionally signed)
u	unsigned decimal, base 10 (input optionally signed)
x, X	unsigned hexadecimal (input optionally signed)
a, A, e, E, f, F, g, G	floating-point number
c	character (with l length modifier, wide character)
s	string (with l length modifier, wide character string)
[matches a sequence of listed characters, ending with]
[^	matches all characters except the ones listed, ending with]
p	pointer to a void
n	pointer to a signed integer into which is written the number of characters read so far
%	a % character
C	wide character (XSI option, equivalent to lc)
S	wide character string (XSI option, equivalent to ls)

Homework

- Esercizio 3: Verificare con la metodologia sperimentale vista a lezione che, in caso di standard I/O, i risultati ottenuti sono quelli di slide 18.