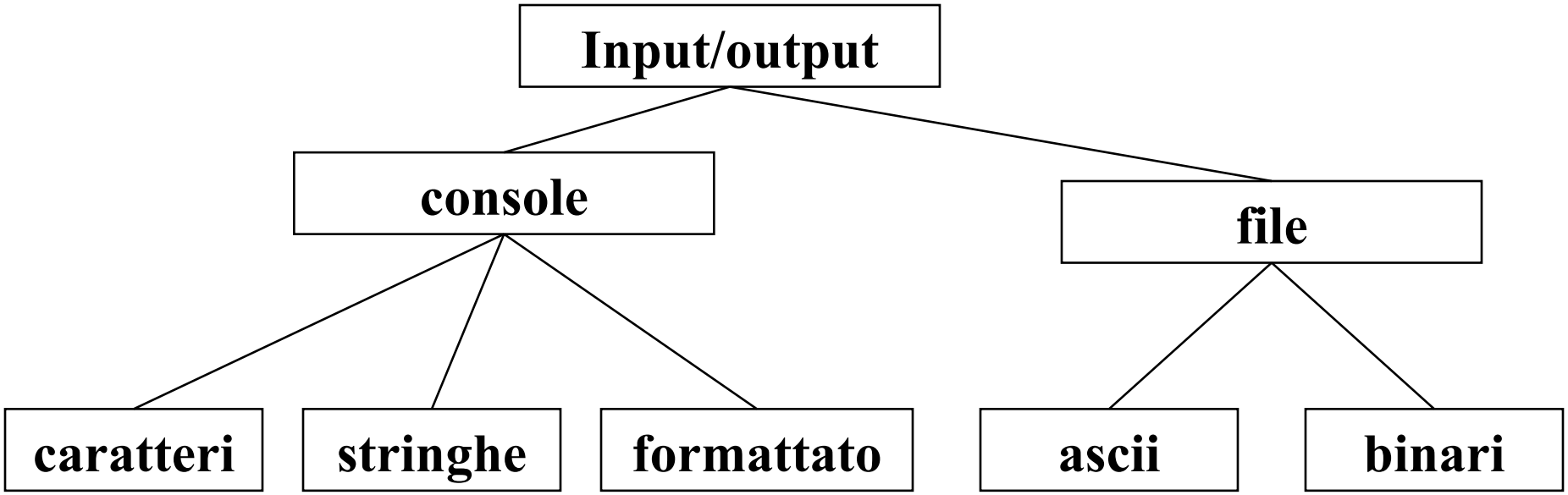
Input/output

Input/Output



La libreria standard del C

- La libreria standard del C è in realtà un insieme di librerie
- Per usare una libreria, non occorre inserirla esplicitamente
- Ogni file sorgente che ne faccia uso deve includere gli header opportuni che contengono le dichiarazioni necessarie

Librerie standard	
input-output	stdio.h
funzioni matematiche	math.h
gestione di stringhe	string.h
operazioni su caratteri	ctype.h
gestione dinamica della memoria	stdlib.h

Il modello di coordinazione del C

- Libreria standard stdio.h
- l'input avviene di norma dal canale standard di input (stdin)
- l'output avviene di norma dal canale standard di output (stdout)
- input e output avvengono sotto forma di una sequenza di caratteri
- tale sequenza è terminata dal carattere speciale EOF (end of file), che varia da una piattaforma ad un'altra
- di norma il canale standard stdin coincide con la tastiera
- di norma il canale standard stdout coincide con il video
- esiste inoltre un altro canale di output, riservato ai messaggi di errore: stderr

Il modello di coordinazione di base

- Poiché sui canali di I/O fluiscono insiemi di caratteri il modello di coordinazione prevede due operazioni base

  scrivere un carattere su un canale di output

  ```
  putchar(ch);
  ```

  leggere un carattere dal canale si input

  ```
  ch = getchar();
  ```

- **Ogni altro tipo di I/O può essere costruito a partire da queste operazioni primitive**

Lettura e Scrittura di Caratteri

int getchar(int *c*);

- stdio.h
- legge un carattere sul canale di intput stdin
- restituisce il carattere letto, EOF nel caso che la sequenza sia finita e in caso di errore
- la stdin è generalmente la tastiera ma potrebbe non esserlo
- la lettura è bufferizzata
- effettua l'echo e attende invio

Lettura e Scrittura di Caratteri

int putchar(int *c*);

- stdio.h
- scrive un carattere sul canale di output (di norma il monitor)
- restituisce il carattere scritto, EOF in caso di errore

Lettura e Scrittura di Caratteri

Non sono standard e quindi non vanno usate

int getch(void);

- conio.h
- ritorna immediatamente ignorando il buffer
- non attende l'invio
- non è standard
- non fa l'eco sullo schermo

```
void printstring(char s[])
{while (*s)
      putchar(*s++);
      putchar('\n');
}

void readstring(char s[])
{int c;
   while ((c=getchar()) != EOF && c !='\n')
     *s++=c;
   *s = '\0';
}
```

Lettura e Scrittura di Stringhe

- **char * gets(char *s);** legge una stringa da tastiera e la memorizza nella locazione indicata nel parametro s. L'ingresso dei dati viene concluso con un CR che viene trasformato in \0
- **int puts(char *s);** visualizza sullo schermo il contenuto dell'argomento di tipo stringa, seguito da un CR. Questa funzione riconosce i caratteri speciali. Il tempo necessario alla sua esecuzione e' inferiore a quello necessario per printf()

Lettura e Scrittura di Stringhe

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

void main( void ) {
    char s[80];

    do {
        gets(s);
        puts(s);
    } while( strlen(s) != 0 );
}
```

I/O di valori numerici

Dato un numero intero calcolare la stringa di caratteri che lo rappresenta e scrivere tale stringa sull'output standard

```
void printInteger(int x){
  char s[20];
  itoa(x,s,10);
  puts(s);
}
```

I/O di valori numerici

Data una stringa calcolare il numero intero che rappresenta

```
void readInteger(int * x){
  char s[20];
  gets(s);
  *x = atoi(s);
}
```

Input/Output Formattato

- Le operazioni di ingresso/uscita avvengono sempre tramite un *formato* che resta sotto il controllo del programmatore.

int printf(*“stringa di controllo”, elencoArgomenti);*

- la stringa di controllo puo’ contenere due tipi di elementi:
  - caratteri che vengono scritti inalterati sullo schermo.
  - direttive di conversione che descrivono il modo in cui gli argomenti devono essere visualizzati. Le direttive di conversione sono costituite da un segno % seguito dal carattere di conversione. printf() consente l’utilizzo di un numero molto grande di specificatori di formato.
- La funzione restituisce il numero di caratteri scritti o un numero negativo in caso di errore

printf(“ora scrivo un intero %d, oppure una %s\n”, 10, “Stringa”);

Input/Output Formattato

%c	singolo carattere	a
%d, %i	intero con segno	-10
%e, %E	notazione scientifica	1e+10
%f	virgola mobile	1.25
%g, %G	il piu' breve fra %e e %f	
%o	ottale senza segno	
%s	stringa di caratteri	
%u	intero senza segno	
%x, %X	esadecimale	xF5
%p	puntatore	
%%	visualizza %	

Modificatori di Formato

- Gli specificatori di formato possono avere varianti che cambiano parzialmente il significato, specificando l'ampiezza dei campi, il numero delle cifre decimali e l'allineamento rispetto al margine sinistro.
- Ampiezza minima del campo: un intero posto fra il segno % e lo specificatore di formato %10d
- Precisione: permette di specificare il numero di cifre decimali in un numero in virgola mobile. %10.3f
- Allineamento: un segno - dopo il segno % fa in modo che il campo venga stampato allineato a sinistra. %-10s
- Il modificatore # fa stampare la variabile di tipo g, f ed e sempre con il punto decimale. Nel caso del formato x premette il prefisso 0x

int scanf(char * stringaDiControllo, elencoArgomenti)

- permette di leggere dati dal dispositivo di ingresso
- Gli specificatori di formato sono analoghi a quelli della funzione printf. La presenza di caratteri diversi da quelli di formato permettono la lettura di questi caratteri senza che vengano memorizzati.
- Gli argomenti di scanf sono **indirizzi**
- E' possibile specificare alcuni modificatori di formato fra cui la massima ampiezza del campo da leggere. Ad esempio scanf("%10s %10s", ...) legge al massimo dieci caratteri per ogni stringa

Gestione dei file

Necessità di persistenza dei file $\Rightarrow$archiviare i file su memoria di massa.

Un file è una astrazione fornita dal sistema operativo, il cui scopo è consentire la memorizzazione di informazioni su memoria di massa

Un file è:

una sequenza di registrazioni (record) uniformi

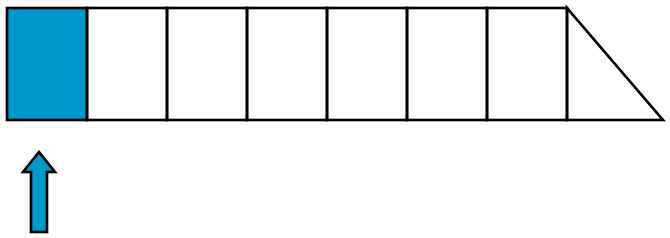
una astrazione di memorizzazione di dimensione potenzialmente illimitata ad accesso sequenziale

CONCETTO DI FILE

Una testina di lettura/scrittura indica in ogni istante il record corrente:

inizialmente la testina si trova per ipotesi sulla prima posizione

dopo ogni operazione di lettura e scrittura la testina si sposta sulla registrazione successiva



E illegale operare oltre la fine del file

CONCETTO DI FILE

A livello di sistema operativo un file è denotato in modo univoco da un nome assoluto che comprende il percorso e il nome relativo.

Esempio:

dos, windows

c:\temp\prova1.c

unix, linux

/usr/temp/prova1.c

Input/Output da file

- Lo standard ANSI definisce un insieme completo di funzioni di I/O su file mentre il vecchio standard conteneva due differenti sistemi di I/O.
- Lo standard ANSI e' basato sul concetto di *stream* che permette di astrarre dal particolare tipo di dispositivo su cui si effettua l'operazione
- Stream di Testo: sono sequenze di caratteri che possono essere terminati da un carattere di NL.
- Stream Binari: sono sequenze di byte con corrispondenza uno a uno con quelli della periferica esterna.
- File: si intende sia un file su disco che un terminale o una periferica di qualsiasi tipo. Ogni file viene associato ad uno stream per mezzo di una struttura di controllo di tipo FILE

Apertura di un File

- per potere usare un file e' necessario aprirlo:

FILE * fopen(const char * nomeFile, const char * modo);

- la funzione apre il file il cui nome e' contenuto della stringa *nomeFile* con il modo *modo*
- la funzione associa uno *stream* e ne restituisce il valore
- Modalità di apertura di un file (*modo*):

"rb"	apre in solo lettura un file binario
"wb"	crea un file per la scrittura un file binario
"ab"	aggiunge ad un file di testo binario
"r+b"	apre un file binario per lettura/scrittura
"w+b"	crea un file binario per lettura/scrittura
"a+b"	aggiunge ad un file binario per lettura/scrittura

Chiusura di un File

- Una volta aperto il file, il programma può operare su esso operando formalmente sulla variabile definita al suo interno. Il sistema operativo provvederà a effettuare realmente le operazioni richieste sul file reale associato a tale simbolo
- Al termine la corrispondenza fra nome del file e variabile usata dal programma per operare su esso dovrà essere soppressa mediante l'operazione di chiusura del file.

Chiusura di un File

- Per chiudere il file si usa la funzione:

  int * fclose(FILE *stream);

- la funzione chiude il file e tutti i buffer ad esso associato saranno cancellati
- la funzione ritorna

  0 se tutto è andato bene

  EOF in caso di errore

FILE BINARI

- Un file binario è una sequenza di byte. Come tale può essere utilizzato per archiviare su memoria di massa qualunque tipo di informazione
- input e output avvengono sotto forma di una sequenza di byte
- la lunghezza del file è registrata dal sistema operativo
- la fine del file e rilevata basandosi sull'esito delle operazioni di lettura
- Poiché un file binario è una sequenza di byte sono fornite due funzioni per leggere e scrivere sequenze di byte

fread e fwrite

Output e input

int fwrite (addr, int dim, int n, FILE *f)

- scrive sul file n elementi ognuno grande dim byte (complessivamente, scrive quindi n*dim byte)
- gli elementi da scrivere vengono prelevati dalla memoria a partire dall'indirizzo addr
- restituisce il numero di elementi (non di bytes) effettivamente scritti che possono essere meno di n

Output e input

int fread (addr, int dim, int n, FILE *f)

- legge da file n elementi ognuno grande dim byte (complessivamente, scrive quindi n*dim byte)
- gli elementi da leggere vengono scritti in memoria a partire dall'indirizzo addr
- restituisce il numero di elementi (non di bytes) effettivamente letti che possono essere meno di n se il file finisce prima. Controllare il valore restituito è il solo modo per sapere se il file è finito

Esempio

Salvare su un file binario (numeri .dat) il contenuto di un array di 10 interi

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{   FILE *fp;
    int vet[10] = {1,2,3,4,5,6,7,8,9,10};
    if ((fp = fopen("numeri.dat", "wb")) == NULL) exit(1);
    fwrite(vet, sizeof(int), 10, fp);
    fclose(fp);
}
```

Leggere da un file binario (numeri .dat) una sequenza di interi e inserirla in un array

```
#include <stdio.h>
#include <stdlib.h>
#define N 10        /* il programma funziona per al più N interi */
main ()
{   FILE *fp;
    int vet[N], i, n;
    if ((fp = fopen("numeri.dat", "rb")) == NULL) exit(1);
    n = fread(vet, sizeof(int), N, fp);
    printf("Il vettore contiene %d elementi", n);
    for (i=0; i<n; i++) printf("%d", vet[i]);
    fclose(fp);
}
```

Output di numeri

L'uso di file binari consente di rendere evidente la differenza di rappresentazione interna di un numero e la sua rappresentazione esterna come stringa di caratteri in una certa base.

int x = 31466

printf("%d", x);

31466

01111010 11101010

fwrite(&x, sizeof(int), 1, stdout);

êz

Input di numeri

int x e digitiamo i caratteri 2 e 3

scanf(“%d”, &x);

23

fread(&x, sizeof(int), 1, stdin);

Scrive dentro x i codici ascii dei due caratteri

FILE DI TESTO

- **E' un caso particolare di file binario che coinvolge una sequenza di caratteri**
- **Ha senso trattarlo come caso a parte perché i caratteri sono un caso estremamente frequente con caratteristiche proprie:**
  - **esiste un concetto di riga e di fine riga (\n)**
  - **certi caratteri sono stampabili a video (quelli di codice >31)**
  - **la sequenza dei caratteri è chiusa dal carattere speciale EOF**
  - **la lunghezza del file è sempre registrata dal sistema operativo (come per ogni file binario) ma è anche registrata dalla presenza del carattere EOF**
- **Quindi, la fine del file può essere rilevata:**
  - **dell'esito delle operazioni di lettura**
  - **perché si intercetta il carattere EOF**

FILE DI TESTO & CANALI I/O

- **I canali di I/O standard non sono altro che file di testo già aperti**
  - **stdin è un file di testo aperto in lettura, di norma agganciato alla tastiera**
  - **stdout è un file di testo aperto in scrittura, di norma agganciato al video**
  - **stderr è un altro file di testo aperto in scrittura, di norma aggangiato al video**
- **Le funzioni di I/O disponibili per i file di testo sono una generalizzazione di quelle già note per i canali di I/O standard**

CONFRONTO

Funzione da console	*Funzione da file*
`int getchar(void);`	`int fgetc(FILE* f);`
`int putchar(int c);`	`int fputc(int c, FILE* f);`
`char* gets(char* s);`	`char* fgets(char* s, int n, FILE* f);`
`int puts(char* s);`	`int fputs(char* s, FILE* f);`
`int printf( … );`	`int fprintf(FILE* f, … );`
`int scanf( … );`	`int fscanf(FILE* f, … );`

- tutte le funzioni da file acquistano una "f" davanti nel nome (qualcuna però cambia leggermente nome)
- tutte le funzioni da file hanno un*parametro in più*, che è appunto il puntatore al`FILE` aperto
  - sempre davanti… tranne in fgets/fputs ☹

fgets() vs. gets() e fputs() vs. puts()

- **fgets() prevede anche un parametro intero n, che consente di leggere non più di n-1 caratteri**
  - **char* fgets(char s[], int n, FILE *f)**
- **E' una caratteristica importante non superare la lunghezza massima della stringa s**
- **A differenza di gets(), che lo elimina, fgets() mantiene il carattere di fine riga, se presente nella stringa letta; aggiunge comunque in fondo il terminatore**
- **Le funzioni di I/O disponibili per i file di testo sono una generalizzazione di quelle già note per i canali di I/O standard**
- **A differenza di puts(), che lo aggiunge sempre, fputs() non inserisce in fondo il carattere di fine**

fgetc() vs. getchar() e fputc() vs. putchar()

- **getchar() e putchar() sono delle scorciatoie linguistiche per fgetc() e fputc()**
  - **getchar() = fgetc(stdin)**
  - **putchar() = fputc(stdout,c)**
- **In effetti, getchar() e putchar() sono quasi sempre delle macro**

Salvare su un file di testo prova.txt ciò che viene battuto sulla tastiera

```
#include <stdio.h>
#include <stdlib.h>
main ()
{     FILE *fp;
      int c;
      if ((fp = fopen("prova.txt", "w")) == NULL) exit(1);
      while ((c=getchar())!=EOF) fputc(c,fp);
      fclose(fp);
}
```

Stampare a video il contenuto di un file di testo prova.txt

```
#include <stdio.h>
#include <stdlib.h>
main ()
{     FILE *fp;
      int c;
      if ((fp = fopen("prova.txt", "r")) == NULL) exit(1);
      while ((c=fgetc(fp))!=EOF) putchar(c);
      fclose(fp);
}
```

fseek() e ftell()

int fseek(FILE *f, long offs, int orig);

long ftell(FILE f);

dove

offs da la posizione rispetto a orig a cui portarsi

orig da la posizione rispetto a cui misurare offs e può essere

SEEK_SET inizio del file

SEEK_CUR posizione corrente

SEEK_END fine file

Scrivere un programma che, dato un file di testo prova.txt, sostituisca tutte le minuscole con maiuscole

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main ()
{     FILE *file;
      char fname[20];
      int ch;
      printf("Inserisci il nome del file");
      scanf("%s", fname);
      if ((file=fopen(fname,"r+"))==NULL) exit(1);
      while ((ch=fgetc(file))!=EOF)
      if (islower(ch))
      {     fseek(file, ftell(file)-1, SEEK_SET);
            fputc(toupper(ch), file);

      }
      fclose(file);
}
```