Laboratorio di informatica

Ingegneria meccanica

Lezione 10 – 10 dicembre 2007

Stringhe in C (1)

- Tipo char per la dichiarazione di variabili carattere (ESEMPIO: char c1 = 'A' ;)
- Tecnicamente si tratta di un tipo intero i cui valori sono rappresentati su singolo byte
- Una stringa è una sequenza di caratteri
- In C, una stringa è un array monodimensionale di elementi di tipo char terminato dal carattere '\0'
- ESEMPIO

  char deg[4] ;

  dichiara deg come una stringa di 3 caratteri utili (il quarto e' il carattere '\0')

Stringhe in C (2)

- Possibile avere dichiarazione con inizializzazione, carattere per carattere, compreso quello di terminazione

  char deg[4] = { 'c' , 'a' , 'p' , '\0' } ;
- Possibile avere inizializzazione con stringa costante con inserimento automatico del carattere '\0' di terminazione

  char deg[4] = "cap" ;
- Nella dichiarazione con inizializzazione, la dimensione della stringa può essere omessa poiché risulta stabilita dalla inizializzazione stessa

  char deg[ ] = "cap" ; /* 4 caratteri in deg */

Stringhe in C (3)

- Possibile accedere in lettura/scrittura agli elementi della stringa utilizzando i loro indici

  deg[1] = 'o' ;

  deg[2] = 'l' ;
- Attenzione a non confondere un singolo carattere 'c' con una stringa di un solo carattere "c" (che contiene 'c' e '\0')
- Una stringa (array di caratteri) può essere stampata con funzione printf specificando %s come specifica di conversione e passando a printf un riferimento all'array di caratteri *sorgente* (nome dell'array o puntatore ad elemento array)

Stringhe in C (4)

- ESEMPIO

```
char nome[ ] = "pippo" ;
char *cPtr ;
cPtr = nome ;
printf( "%s" , nome ) ; /* stampa pippo */
printf( "%s" , cPtr ) ; /* stampa pippo */
cPtr++ ;
printf( "%s" , cPtr ) ; /* stampa ippo */
```


Stringhe in C (5)

- Una stringa può essere acquisita con funzione scanf specificando %s come specifica di conversione e passando a scanf un riferimento all'array di caratteri *destinazione* (nome dell'array o puntatore ad elemento array)
- scanf acquisisce la sequenza di caratteri da stdin fino al primo carattere *spazio vuoto* escluso (spazio vuoto include: spazio, tabulazione, fine linea) e completa la sequenza con il carattere '\0' di terminazione
- Nel dimensionare un array di caratteri (da usare come *stringa*) occorre considerare il carattere extra di terminazione ('\0')

Stringhe in C (6)

- ESEMPIO

```
char nome[ 20 ] ;
scanf( "%s" , nome ) ; /* inserito pippo pippo */
printf( "%s" , nome ) ; /* stampa pippo */
```

- ATTENZIONE

  Tipicamente, funzione scanf acquisisce sequenza caratteri da convertire nei valori desiderati (qualunque tipo di variabile) non direttamente da stdin ma da un *buffer* (area di memoria in cui vengono registrati tutti i caratteri inseriti da stdin)

scanf

- Non sempre i caratteri inseriti sono tutti utilizzati per la conversione specificata. I caratteri rimanenti vengono automaticamente usati nella scanf successiva (se compatibili)
- ESEMPIO

```
char nome[ 20 ] ;
scanf( "%s" , nome ) ; /* inserito pippo pippo */
printf( "%s" , nome ) ; /* stampa pippo */
```

- Una successiva scanf troverebbe disponibile nel buffer la sequenza di caratteri " pippo"
- *Si può forzare lo svuotamento del buffer di input con l'istruzione fflush(stdin) ;*

File (1)

- In un sistema di elaborazione dati, la memorizzazione *permanente* di dati richiede l'uso di memoria *non volatile* (memoria secondaria)
- Ad alto livello (programmazione in linguaggi di alto livello e/o uso di software applicativo), la memoria secondaria viene vista—grazie al sistema operativo—come una collezione di *file*, organizzabili secondo una gerarchia di livelli
- Ad un dato livello della gerarchia si trovano file e/o collezioni di file, dette *directory*, contenenti file o altre directory
- Un file è una collezione di dati

File (2)

- Un file su memoria secondaria è univocamente individuato da nome e livello occupato nella gerarchia di memoria
- Alcune tipiche operazioni su file: creazione, duplicazione, eliminazione, scrittura, lettura, modifica, rinomina, (*configurazione diritti di accesso*)
- Il risultato di un'elaborazione scritto su file può diventare l'input per una nuova elaborazione
- *Memorizzazione permanente* è da intendersi con la dovuta cautela: si può avere *perdita di dati* per guasti di vario tipo. Il rischio di perdita può essere ridotto con l'uso di copie (di back-up) memorizzate su supporti di memoria distinti

I file in C (1)

- In C, input ed output di dati sono modellati come flussi di I/O (I/O stream)
- Un flusso è una sequenza ordinata di byte
- Esistono tre flussi standard: stdin, stdout e stderr (stdin e stdout già incontrati: tipicamente associati a tastiera e terminale video, rispettivamente)
- *stderr è un flusso di output tipicamente connesso a terminale video, utilizzabile per la comunicazione di avvisi/errori*
- Un flusso di I/O diventa utilizzabile dopo la sua *apertura* e può essere usato fino alla sua *chiusura* (apertura e chiusura di stdin, stdout, stderr avvengono in modo automatico)

I file in C (2)

- L'uso di un file richiede la sua associazione ad un flusso
- L'associazione tra un file ed un flusso è stabilita con l'operazione di apertura del file. Tale associazione viene rimossa con una corrispondente operazione di chiusura del file
- *In questo corso*, ci si limita a considerare file di testo (text file) ovvero, file associati a flussi di testo (text stream)
- Un file di testo può essere organizzato come una sequenza di linee di testo, con ogni linea costituita da una sequenza di caratteri (eventualmente vuota) terminata da un carattere di *newline* (\n)

Apertura di un file in C

- Avviene con funzione fopen specificando un nomefile ed una modalità
- nomefile può essere una stringa indicante il nome del file
- Esistono varie modalità specificabili con stringhe opportune, tra cui "w" per *scrittura di testo* ed "a" per *aggiunta di testo*
- fopen restituisce un puntatore ad un tipo speciale detto FILE
- Il puntatore restituito da fopen diventa il riferimento per il file aperto ed è utilizzato nelle operazioni su quel file

fopen

- In caso di errore (impossibilità di apertura file), fopen restituisce un puntatore di valore NULL
- L'apertura in modalità "w"/"a" di un *nuovo* file provoca la creazione di questo file e ne rende possibile la scrittura
- L'apertura in modalità "w" di un file *già esistente* provoca la perdita del contenuto precedente del file
- L'apertura in modalità "a" di file già esistente, provoca l'aggiunta di testo in coda al contenuto precedente del file

Esempi d'uso di fopen

```
FILE *fPtr ; /* fPtr puntatore a tipo FILE */

ES1: fPtr = fopen( "prova" , "w" );
    if ( fptr == NULL )
            printf( "errore apertura file" ) ;
      else { … }

ES2: if (( fPtr = fopen( "prova" , "w" )) != NULL )
          { … }
       else printf( "errore apertura file" ) ;

/* errore: messaggio opportuno */
/* assenza errore: apertura (stream associato a)
 file "prova" per scrittura (testo) */
```


Scrittura file con fprintf

- Funzione fprintf esegue su file le operazioni che printf esegue su stdout
- L'uso di fprintf richiede di specificare un puntatore associato al flusso di output

- ESEMPIO

```
FILE *fPtr ; /* fPtr puntatore a tipo FILE */
int a=3 , b=5 ;

if (( fPtr = fopen( "prova" , "w" )) == NULL )
    printf( "errore apertura file" ) ;
else fprintf( fPtr , "a*b = %d\n" , a*b ) ;
```

Nome file in C (1)

- Funzione fopen con argomenti nomefile e modalità
- nomefile può essere:

1. una *stringa costante*, come in

```
FILE *fPtr ;
if (( fPtr = fopen( "A:\\prova" , "w" )) == NULL )
...
```

2. *un array di caratteri contenente una sequenza terminata dal simbolo speciale '\0'*, come in

```
FILE *fPtr ;
char name[ 21 ] ;
scanf( "%s" , name ) ; /* inseriti max 20 caratteri */
if (( fPtr = fopen( name , "w" )) == NULL )
...
```


Nome file in C (2)

3. *un puntatore a tipo char che punti al primo carattere di una sequenza terminata con simbolo speciale '\0'*, come in

```
FILE *fPtr ;
char str[ ] = "fname" , *fnPtr = str ;
fnPtr++ ;
if (( fPtr = fopen( fnPtr , "w" )) == NULL )
...
```


Nome file (1)

- Il nome del file permette di individuarlo univocamente nella gerarchia di file associata ad un dispositivo di memoria secondaria
- `fPtr = fopen( "prova" , "w" )`
  provoca la creazione/apertura del file "prova" nella directory contenente il *file eseguibile* del programma (*directory corrente* specificata implicitamente)
- Possibile specificare una diversa posizione in cui cercare/creare un file fornendo nel nome del file le informazioni necessarie (relative alla directory corrente o assolute)

Nome file (2)

- Esiste limite per numero di caratteri in nome file: FILENAME_MAX in stdio.h
- Esempio di specifica di *percorso file* per sistema Windows

  `fPtr = fopen( "A:\\prova" , "w" )`

  provoca la creazione/apertura del file "prova" nel *livello radice* del disco floppy

  `fPtr = fopen( "A:\\esempi\\prova" , "w" )`

  provoca la creazione/apertura del file "prova" nella directory "esempi" del disco floppy (directory "esempi" nel *livello radice*) (directory "esempi" DEVE già esistere)

fclose

- Funzione fclose esegue la chiusura di file precedentemente aperto con funzione fopen (si tratta della chiusura del flusso associato al file)
- Il mancato uso di fclose può provocare la perdita di dati (*reale scrittura su file potrebbe essere attivata dalla operazione di chiusura*) ed altri problemi (*esiste un limite al numero di file contemporaneamente aperti: FOPEN_MAX in stdio.h*)
- fclose richiede di specificare il puntatore associato al file da chiudere
- fclose(puntatoreFILE)

Esempio uso file (1)

```
…     /* file "prova" non preesistente */
FILE *fPtr ; /* fPtr puntatore a tipo FILE */
long int k ;
int max = 1000 ;
…
if (( fPtr = fopen( "prova" , "w" )) == NULL )
   printf( "errore apertura file" ) ;
else {
   fprintf( fPtr , "tabella quadrati\n\n" ) ;
   for( k = 0 ; k < max ; k++ )
   fprintf( fPtr , "%ld\t\t%ld\n" , k , k*k ) ;
   fclose( fPtr ) ;
}
…
```


Esempio uso file (2)

```
……    /* file "prova" preesistente */
FILE *fPtr ; /* fPtr puntatore a tipo FILE */
long int k ;
int max = 1000 ;
char nomefile[ ]= "prova";
…
if (( fPtr = fopen( nomefile , "a" )) == NULL )
   printf( "errore apertura file" ) ;
else {
   fprintf( fPtr , "tabella cubi\n\n" ) ;
   for( k = 0 ; k < max ; k++ )
   fprintf( fPtr , "%ld\t\t%ld\n" , k , k*k*k ) ;
   fclose( fPtr ) ;
}
…
```


Lettura di un file in C

- L'apertura in modalità "r" di un file *già esistente* ne rende possibile la lettura
- ESEMPIO

```
…
if (( fPtr = fopen( "prova" , "r" )) == NULL )
    printf( "errore apertura file" ) ;
else {  /* possibile accesso in lettura al file */
 …
     fclose( fPtr ) ;
}
…
```

Lettura di un file (1)

- Un file testo può essere letto carattere per carattere, dal primo all'ultimo (organizzazione tipica: sequenza di linee/righe di testo, con ogni linea costituita da una sequenza di caratteri (eventualmente vuota) terminata da un carattere di newline. Un file può essere *vuoto*
- A basso livello, l'accesso a file è gestito dal sistema operativo (le funzioni viste ad alto livello *interagiscono* con il sistema operativo)
- Le funzioni di alto livello per lettura file riportano tipicamente al programma informazioni sullo stato del file (possibile sapere se nel file ci sono ancora caratteri da leggere)

Lettura di un file (2)

- La lettura di un file carattere per carattere (lettura non formattata) non tiene conto di un eventuale *formato di scrittura* presente nel file
- Se il file è stato *scritto* imponendo alle sue linee una determinata struttura, allora il file può tipicamente essere *letto* tenendo conto di tale struttura (formato)
- Descrivere il formato di un file da leggere significa specificare quanto necessario per impostare una sua lettura formattata
- ESEMPIO: *Il file contiene dieci linee; ogni linea contiene due stringhe di max 12 caratteri separate da un carattere speciale; etc.*

C: Funzione fscanf (1)

- *Funzione fprintf esegue su file le operazioni che printf esegue su stdout*
- *L'uso di fprintf richiede di specificare un puntatore associato al flusso di output*
- fprintf(stdout , …) equivalente a printf(…)
- Funzione fscanf esegue su file le operazioni che scanf esegue su *stdin*
- L'uso di fscanf richiede di specificare un puntatore associato al flusso di input
- fscanf(stdin , …) *equivalente* a scanf(…)

C: Funzione fscanf (2)

- Funzione fscanf può essere usata per la lettura formattata di un file di testo
- Specificando l'acquisizione di un singolo carattere si può eseguire una lettura non formattata
- Attenzione: 1. Leggere un file testo *carattere per carattere* è sempre possibile, indipendentemente dal formato del file; 2. Leggere un file testo assumendo un certo formato richiede che il file sia realmente compatibile con quel formato
- A lettura ultimata (non ci sono più caratteri da leggere nel file testo), fscanf restituisce il valore speciale EOF (definito in stdio.h; valore tipico -1)

Esempio: conteggio caratteri in un file

```
#include <stdio.h>
int main( ) {
   FILE *fPtr ;
   char ch ;
   long int cont = 0 ;
   if (( fPtr = fopen( "provar" , "r" )) != NULL ) {
          while( fscanf( fPtr , "%c" , &ch ) != EOF ) {
              cont++ ; }
          printf( "cont = %ld\n" , cont ) ;
          /* cont = numero caratteri in "provar" */
          fclose( fPtr ) ;
   }
   else printf( "errore apertura file" ) ;
   return 0;
}
```


Esempio: copia di un file

```
…
FILE *frPtr , *fwPtr;
char nfr[21] , nfw[21] , ch ;
…
if ((( frPtr = fopen( nfr , "r" )) != NULL ) &&
    (( fwPtr = fopen( nfw , "w" )) != NULL )) {
       while( fscanf( frPtr , "%c" , &ch ) != EOF )
              fprintf( fwPtr , "%c" , ch ) ;
       fclose( frPtr ) ;
       fclose( fwPtr ) ; }
else printf( "errore apertura file" ) ;
…
```


Esempio: acquisizione array da file (1)

```
…
FILE *frPtr ;
int v[ max_d ] , pos ;
…
/* Formato "comp.dat": 1 int per linea; numero linee =
 max_d */

if ((( frPtr = fopen( "comp.dat" , "r" )) != NULL )) {
    for( pos = 0 ; pos < max_d ; pos++ )
       fscanf( frPtr , "%d" , &v[ pos ] ) ;
 …
    fclose( frPtr ) ;
}
else printf( "errore apertura file" ) ;
   …
```


Esempio: acquisizione array da file (2)

```
…
FILE *frPtr ;
int v[ max_d ] , pos = 0 ;
…
/* Formato "comp.dat": 1 int per linea; max numero
    linee = max_d */

if ((( frPtr = fopen( "comp.dat" , "r" )) != NULL )) {
    while( fscanf( frPtr , "%d" , &v[ pos ] ) != EOF )
       pos++ ;
    fclose( frPtr ) ;
}
else printf( "errore apertura file" ) ;
   …
```

Esempio: acquisizione array bidimensionale

```
/* Formato "comp2.dat": d2 int per linea (separati da
spazio/tab); d1 linee */
...
FILE *frPtr ;
int v[ d1 ][ d2 ] , r , c ;
...
if ((( frPtr = fopen( "comp2.dat" , "r" )) != NULL )) {
    for( r = 0 ; r < d1 ; r++ )
       for( c = 0 ; c < d2 ; c++ )
          fscanf( frPtr , "%d" , &v[ r ][ c ] ) ;
    ...
    fclose( frPtr ) ;
}
else printf( "errore apertura file" ) ;
   ...
```