Esercitazione di Reti degli elaboratori

Prof.ssa Chiara Petrioli

SAPIENZA
UNIVERSITÀ DI ROMA

Laboratorio di C

Luca Iezzi
iezzi@diag.uniroma1.it

Argomenti

- Altri utilizzi dei puntatori
- Allocazione dinamica della memoria
- File

Esercizio 1

Si scriva un programma che consente all’utente di inserire 5 numeri interi e li salva in un array.

Il programma deve, tramite l’utilizzo di una funzione (chiamata *funzione 1*), calcolare la somma degli elementi, trovare il minimo e il massimo.

Alcuni problemi...

Se l'utente necessita di inserire più valori?

È opportuno modificare il codice sorgente aumentando la dimensione dell'array e quindi compilare nuovamente il programma.

Se invece vogliamo consentire all'utente di scegliere il numero di elementi da inserire?

Potremmo definire un array molto grande (ad esempio 100 elementi) e consentirgli di inserire numeri fino a che non ne inserisce uno negativo...

Esempio...

Esempio

```
1   #include <stdio.h>
2
3   int funzione1(int array[], int size, int *min, int *max);
4
5   int main(){
6       int ARRAY_SIZE = 100;
7       int arr[ARRAY_SIZE];
8       int size = 0;
9       printf("Quanti elementi si vogliono inserire?\n");
10      scanf("%d",&size);
11      if ((size < 0) || (size > 100)){
12          printf("Scelta non consentita\n");
13          return 0;
14      }
15      for (int i=0;i<size;i++){
16          printf("Inserisci elemento %d: \n",i+1);
17          scanf("%d",&arr[i]);
18      }
19      int min = 0, max = 0;
20      int somma = funzione1(arr, size, &min, &max);
21      printf("SOMMA: %d - MIN: %d - MAX: %d \n\n",somma, min, max);
22      return 0;
23  }
```

N.B. *funzione1* è identica all'esercizio precedente.....

Alcuni problemi....

Se l'utente inserisce pochi elementi?

Abbiamo sicuramente uno spreco di memoria...

Se l'utente ne vuole inserire più di 100?

È opportuno modificare e compilare nuovamente il programma...

O ancora …

Alcuni problemi....(2)

Supponiamo di voler scrivere una funzione:

char **dupstr(**char** *s)*

che ritorna una copia della stringa di input s.
Potremmo scriverla così?

```
char *dupstr(char *s)
{
    char dup[strlen(s) + 1];
    strcpy(dup, s);
    return dup;
}
```

Allocazione dinamica della memoria

- **Risposta**: **No!**
- **dup è un array locale!**

- La memoria riservata per la variabile locale esiste solo all'interno della funzione che la dichiara.

- Per scrivere una funzione di questo tipo dobbiamo ***allocare memoria il modo dinamico***, cioè riservare locazioni di memoria per la durata dell'esecuzione del programma (*o passare l'array da ritornare nei parametri di dupstr*).

- **Inoltre**, la dimensione dell'array non è nota a priori (durante la compilazione), quindi non possiamo semplicemente allocare la memoria in modo statico ma dobbiamo allocare una dimensione di memoria determinata durante l'esecuzione del programma.

Alcuni problemi....

Come è possibile vedere, l’allocazione statica della memoria ha dei grossi limiti!

Per una maggiore flessibilità, è opportuno utilizzare l’allocazione dinamica.

Allocazione dinamica della memoria

- La memoria è divisa sostanzialmente in due parti:

Tipo Memoria	Descrizione
Stack	È una parte **statica**, che contiene tutto ciò che sappiamo sarà allocato di certo (tutto ciò che è dichiarato nel codice quindi, come una variabile int o un array allocato staticamente come abbiamo visto nelle lezioni precedenti)
Heap	È una parte **dinamica**, in cui la dimensione degli elementi può cambiare a “runtime” ovvero durante l’esecuzione del programma

Allocazione dinamica della memoria

- Il C supporta l'allocazione dinamica della memoria attraverso l'uso delle funzioni definite nella libreria **stdlib.h**
- Le principali funzioni sono:

Funzione	Descrizione
void* malloc(size)	riserva una buffer di size bytes e ne ritorna l'indirizzo.
void* calloc(size)	Come la malloc, solamente che inizializza anche la memoria con tutti 0.
void* realloc(void* ptr, size)	realloca la memoria puntata da ptr alla dimensione size bytes, o realloca la memoria se size = 0
void free(void* ptr)	libera la memoria riservata da malloc, calloc o realloc

Allocazione dinamica della memoria

- Queste funzioni ritornano o accettano come argomenti puntatori di tipo **void*** □ indica un puntatore ad un blocco di memoria generico.

- Se necessario, questo tipo è automaticamente convertito ad altri tipi puntatore.

Allocazione dinamica della memoria

- Per determinare la quantità di bytes necessaria per i vari tipi si usa l'operatore
  - **sizeof**(<tipo>)
  - **sizeof**(<variabile>)

- Ad esempio:
  - **sizeof**(char) ritorna 1;
  - **sizeof**(float) ritorna 4

Allocazione dinamica della memoria

- Invece, per una variabile *v* di un tipo *x*
  □ **sizeof**(v) ritorna **sizeof**(*x*). Ad esempio, sia v di tipo **int**:
  - **sizeof**(*v*) = **sizeof**(*int*)

- Nel caso di tipi array,
  l'operatore **sizeof**(<tipo>[<dimensione>]) è uguale a **sizeof**(<tipo>)*<dimensione>.

Allocazione dinamica della memoria

Adesso possiamo scrivere la versione corretta della nostra funzione dupstr():

```
char *dupstr(char *s) {    //ritorna una copia della stringa s
    int n = strlen(s);
    char *dup = malloc((n + 1)*sizeof(char));
    if (dup == NULL) {
        //allocazione fallita
    }
    strcpy(dup, s);
    return dup;
}
```

Esercizio 2

Si svolga nuovamente l’esercizio 1 utilizzando un array allocato dinamicamente.

Soluzione (1/2)

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

int funzione1(int array[], int size, int *min, int *max);

int main(){
    int ARRAY_SIZE = 0;
    printf("Quanti elementi si vogliono inserire?\n");
    scanf("%d",&ARRAY_SIZE);
    int *array = (int*)malloc(ARRAY_SIZE * sizeof(int));
    if (array == NULL){
        printf("Non è possibile allocare memoria!\n");
        return 0;
    }
    for (int i=0;i<ARRAY_SIZE;i++){
        printf("Inserisci elemento %d: \n",i+1);
        scanf("%d",&array[i]);
    }
    int min = 0, max = 0;
    int somma = funzione1(array, ARRAY_SIZE, &min, &max);
    printf("SOMMA: %d - MIN: %d - MAX: %d \n",somma, min, max);
    free(array);
    printf("Memoria liberata correttamente!\n");
    return 0;
}
```

Soluzione (2/2)

```
26
27  int funzione1(int array[], int size, int *min, int *max){
28      if (size == 0){
29          *max = 0;
30          *min = 0;
31          return 0;
32      }
33      int somma = 0;
34      *min = array[0];
35      *max = array[0];
36      for(int i=0;i<size;i++){
37          somma += array[i];
38          if (*min > array[i])
39              *min = array[i];
40          if (*max < array[i])
41              *max = array[i];
42      }
43      return somma;
44  }
```

Esercizio 3

Scrivere la funzione ***int*** **occ(**int** A[], **int** n, **int** x, **int** *nocc)*
che ritorna in un array, allocato dinamicamente, le posizioni dell'array *A* che contengono il valore *x* e in **nocc* restituisce il numero di tali posizioni.

Esercizio 4

Si scriva una funzione ***char*** **inputstr()* che ritorna in una stringa, allocata dinamicamente, la sequenza di caratteri letti dall'input (fino a '\n').

Esercizio 5

Scrivere una funzione

int **mulArrayD(**int** A[], **int** n)*

che ritorna un array, allocato dinamicamente, con tutti gli elementi dell'array A moltiplicati per 2.

Esercizio 6

Scrivere una funzione

char **concat(**char** *s1, **char** *s2)*

che ritorna una nuova stringa che contiene la concatenazione delle stringhe s1 e s2. Ad esempio, se *s1* è "Prima parte" e *s2* è "Seconda parte" allora la funzione ritorna la stringa "Prima parteSeconda parte".

Esercizio 7

Scrivere un programma che consenta all'utente di inserire in input due stringhe che devono essere memorizzate in due array allocati dinamicamente.

Il programma deve memorizzare in un terzo array (sempre allocato dinamicamente) la concatenazione delle due stringhe.

N.B. **Non utilizzare le funzioni definite nella libreria string.h.**

Esercizio 8

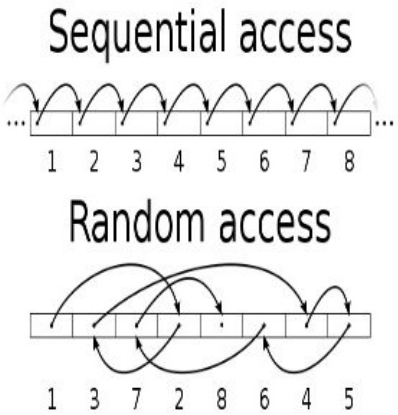
Scrivere lo stesso programma dell’esercizio precedente, ma utilizzando le funzioni nella libreria string.h

File I/O

- I file mantengono nella memoria di un computer qualunque tipo di informazioni: testi, dati, immagini, video, musica, ecc.
- Il C mette a disposizione alcune funzioni per creare, leggere e scrivere sui File.

File I/O

- Vediamo come gestire i dati e le informazioni presenti in un file in C.
- Tipi di file:
  - accesso sequenziale (file di testo).
  - accesso casuale (file binari).



Gestione dei file

Apertura del file

Al fine di accedere ad un file è opportuno aprirlo. Questa operazione inizializza un puntatore al file.

Accesso al file

Una volta aperto, è possibile leggere/scrivere/modificare tramite il puntatore.

Chiusura del file

Una volta terminate le operazioni è opportuno chiudere il file per memorizzare permanentemente il suo contenuto.

Apertura del file

FILE fopen(char *fileName, char *mode);

fileName: array di caratteri che rappresenta il nome (relativo o assoluto) del file che si desidera aprire.

mode: indica la modalità di apertura del file

- "w": scrittura (se il file esiste lo tronca, altrimenti lo crea)
- "a": scrittura in append: scrive a partire dalla fine del file (se non esiste lo crea)
- "r+": lettura e scrittura file esistente (se non esiste ritorna NULL)
- "w+": lettura e scrittura (se il file esiste lo tronca, altrimenti lo crea)
- "a+": lettura e scrittura in append: scrive (e legge) a partire dalla fine del file (se non esiste lo crea)

Se l'operazione di apertura va a buon fine, la funzione restituisce un riferimento al file aperto, altrimenti restituisce NULL (se ad esempio il file non esiste).

Esempio di apertura di un file

apertura di un file in lettura:

FILE *filePtr = fopen("file1.txt", "r");

apertura di un file in scrittura:

FILE *filePtr = fopen("file1.txt", "w");

Se il file esiste il suo contenuto viene perso, altrimenti ne viene creato uno nuovo.

Chiusura di un file

Al termine delle operazioni, è opportuno chiudere il file:

int fclose(FILE *file);

Il valore di ritorno è 0 se il file è stato chiuso correttamente, la costante EOF altrimenti.

Un esempio...

```
2   FILE *fp = fopen("file.txt","w"); //apertura file
3   if(fp == NULL) {
4       printf("Errore apertura file...");
5       //...gestione dell'errore....
6   }
7   //operazioni di scrittura......
8   fclose(fp); //chiusura file
```

File I/O

Alcune funzioni per leggere/scrivere i file

- ***void*** *rewind(**FILE** *f)*: riporta il cursore di f all'inizio del file.
- ***int*** *fgetc(**FILE** *f)*: ritorna il prossimo carattere (byte) del file f e avanza il cursore. Se il cursore è alla fine del file o si verifica un errore ritorna EOF.
- ***int*** *fputc(**int** c, **FILE** *f)*: scrive il carattere c nella posizione del cursore del file f, avanza il cursore e ritorna il carattere scritto. Se si verifica un errore, ritorna EOF.

File I/O

Funzioni principali in stdio.h

- ***int*** *fscanf(**FILE** *f, **const char** *format, ...)*: del tutto simile alla scanf, solo che legge dal file f invece che dallo standard input. Ritorna il numero di assegnamenti effettuati.
- ***int*** *fprintf(**FILE** *f, **const char** *format, ...)*: del tutto simile alla printf, solo che scrive nel file f anziché nello standard output. Ritorna (in quasi tutte le implementazioni) il numero di caratteri scritti, in caso di errore ritorna un valore negativo.

File I/O

Funzioni principali in stdio.h

- ***char*** **fgets(**char** *s, **int** n, **FILE** *f)*: legge una linea dal file f e la copia in s.
- Legge e copia i caratteri nel blocco puntato da s finché incontra:
  - un ‘\n’
  - la fine del file
  - ha letto n - 1 caratteri.
- La stringa s è sempre terminata da '\0’.
- Se incontra la fine del file senza aver letto alcun carattere ritorna NULL(e s rimane invariata), altrimenti ritorna s.

File I/O
Esempio

Programma che prende come argomento del main il nome di un file (di testo) e permette all'utente di stampare a video le linee del file pagina per pagina (uso di fopen(), fscanf()e fclose()).

File I/O
Esempio

Programma che legge un file e ne stampa il contenuto.

```
#include <stdio.h>

#define MAX_LINE_SIZE 80

int main()
{
    char buf[MAX_LINE_SIZE];
    FILE *fp;
    fp=fopen("testo.txt", "r");
    if(fp!=NULL) {
        while (fscanf(fp,"%s",buf)>0)
            printf("%s\n", buf);
        fclose(fp);
    } else {
        printf("Error opening the file\n");
    }
    return 0;
}
```

File I/O
Esempio

Programma che scrive in un file la lista della spesa.

File I/O

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

#define MAX_LINE_SIZE 80

int main()
{
    //scrittura in un file della lista della spesa
    FILE *fp;
    char item[MAX_LINE_SIZE];
    int fine=0;
    fp=fopen("testo.txt", "w");
    if( fp != NULL) {
        printf(" Lista della spesa: ");
        do
        {
            printf("\nprossimo articolo ? ");
            scanf("%s", item);
            fprintf(fp, "%s\n", item);
            printf("\nFinito (si=1, no=0)? ");
            scanf("%d", &fine);
        } while (!fine);
        fclose(fp);
        printf("Fine scrittura\n");
    } else {
        printf("Errore nell'apertura del file\n");
    }
    return 0;
}
```

Esercizio 9

Esercizio. Scrivere un programma che prende come input i nomi di due file di testo e appende il contenuto del secondo file al primo file. Ad esempio, se il primo file contiene *hello* e il secondo file world, allora il programma modifica il primo file con il testo *helloworld*.

Per provare l'esercizio creare due file file1.txt contenente la stringa "hello" e file2.txt contenente la stringa "world".

Esercizio 10

Scrivere un programma che legge in input il nome di un file che ad ogni riga contiene:

nomeStudente cognomeStudente voto

Il programma deve stampare sullo schermo, per ogni studente, il nome, il cognome e il voto dell'esame.

Alla fine inoltre, deve stampare il voto più alto, quello più basso e la media aritmetica dei voti.

N.B. Si assuma che il file sia formattato correttamente

Esercizio 11

Scrivere un programma che legge in input il nome di un file.

Il programma deve leggere il contenuto del file e memorizzarlo all’interno di un array allocato dinamicamente.

Esercizio 12

Si scriva un programma che legge in input il nome di un file e una stringa S.

Il programma deve stampare a video, quante volte compare la stringa S all’interno del file