LABORATORIO DI PROGRAMMAZIONE 1
CORSO DI LAUREA IN MATEMATICA
UNIVERSITÀ DEGLI STUDI DI MILANO
2019–2020

INDICE

Ultima revisione: 21 febbraio 2020.

Parte 1. Allocazione dinamica della memoria

In questa prima parte della lezione, per allocare dinamicamente la memoria userete le funzioni `malloc`, `calloc`, `realloc` e `free` dichiarate nel file di intestazione `stdlib.h`.

Esercizio 1

Ordinamento BubbleSort *con allocazione dinamica.*

Tempo: 25 min.

Si scriva un programma che legga dalla console una successione di valori `double`, e li memorizzi in una zona della memoria di dimensione appropriata. Chiedete all'utente per prima cosa quanti valori `double` intenda inserire. Usate poi questa informazione per eseguire un'appropriata chiamata alla funzione `malloc`. Dopo aver letto e memorizzato i valori, il programma li ordina applicando l'algoritmo BubbleSort — si veda l'Esercizio 11 della Lezione 4 — visualizza l'elenco ordinato, libera la memoria allocata dinamicamente e termina.

Esercizio 2

Ordinamento BubbleSort *con allocazione dinamica, menu, e funzioni.*

Tempo: 35 min.

Si scriva un programma che visualizzi un menu come segue:

```
1. Inserisci elenco di double.
2. Ordina elenco.
3. Visualizza elenco.
4. Termina.
```

Se l'utente sceglie `4` il programma termina. Se l'utente sceglie `2` o `3` prima di aver inserito un elenco di `double`, il programma informa l'utente della necessità di inserire dei dati e torna al menu. Se l'utente sceglie `1` il programma permette all'utente di inserire un elenco di valori di tipo `double`, e li memorizza usando le funzioni di `stdlib.h` per l'allocazione dinamica della memoria. Chiedete preliminarmente all'utente quanti valori intende inserire, come nell'Esercizio 1. Stabilite un comportamento sensato del programma nel caso in cui l'utente scelga `1` e inserisca, o tenti di inserire, zero valori. L'opzione `3` permette di visualizzare l'elenco corrente. L'opzione `2` ordina l'elenco corrente applicando l'algoritmo BubbleSort. Se l'utente sceglie `1` dopo aver già inserito un elenco, il programma scarta il vecchio elenco (deallocando la memoria con `free`) e permette all'utente di inserirne uno nuovo, allocando la memoria necessaria di conseguenza. Strutturate il programma in funzioni appropriate.

Esercizio 3

Ordinamento BubbleSort *con riallocazione dinamica.*

Tempo: 30 min.

Si modifichi il programma scritto per risolvere l'Esercizio 2 di modo che il menu diventi:

```
1. Inserisci elenco di double.
2. Ordina elenco.
3. Visualizza elenco.
4. Aggiungi elementi.
5. Termina.
```

L'opzione `4` permette all'utente di aggiungere valori `double` in coda all'elenco di valori corrente. Chiedete preliminarmente all'utente quanti valori intende aggiungere all'elenco corrente. Usate `realloc` per ridimensionare la memoria allocata dinamicamente. Se l'utente sceglie `4` prima di aver inserito un elenco di `double` tramite `1`, il programma informa l'utente della necessità di inserire dei dati e torna al menu. Per il resto il funzionamento del programma è come nell'Esercizio 2.

Esercizio 4

Ordinamento di `double` *su file*.

Tempo: 30 min.

Scrivete un programma che riceva due nomi di file dalla riga di comando. Il programma apre il primo file in lettura, e assume che esso sia formattato come un elenco di valori `double`, scritti uno per riga. Il programma legge i valori `double` usando la funzione `fscanf` di `stdio.h`, e li memorizza usando l'allocazione dinamica della memoria. Scrivete a tal fine una funzione `add`, di prototipo appropriato, che esegua il compito di aggiungere alla memoria ciascun nuovo valore `double` passato in argomento. La funzione dovrà usare `realloc`. Il programma esegue poi un ordinamento dell'elenco di numeri, usando BubbleSort. Il programma infine scrive sul secondo file ricevuto dalla riga di comando la lista ordinata, usando `fprintf` di `stdio.h`. Prima di terminare, il programma visualizza il contenuto di questo secondo file sul terminale.

Parte 2. Ripasso: Array e programmazione strutturata

In questa seconda parte della lezione scriverete un singolo programma, sviluppandolo però un po' alla volta come indicato dai diversi esercizi. Questa è la struttura che hanno i temi d'esame, anche se l'esempio che vedrete qui è più semplice di un tema d'esame tipico. Gli argomenti coinvolti sono gli array bidimensionali, le variabili globali e la programmazione strutturata. Cercate di completare questa seconda parte della lezione in un'ora di tempo circa.

Esercizio 5

La funzione `shift`.

In questo esercizio e nei successivi considereremo array bidimensionali di caratteri che permettano di rappresentare matrici rettangolari come quella che segue:

```
+ - -
+ + +
- - -
```

Ciascun elemento della matrice è o il carattere `+`, o il carattere `-`. La matrice ha nome `mat` e ha 3 righe e 3 colonne. Dichiaratela come variabile globale del tipo appropriato. Implementate una funzione `shift`, di prototipo appropriato, che faccia scorrere in avanti le colonne della matrice di un passo; l'ultima colonna diventa la prima. Per esempio, dopo l'invocazione della funzione, la matrice sopra raffigurata diviene:

```
- + -
+ + +
- - -
```

Prima di procedere, assicuratevi di aver testato il buon funzionamento di `shift`. A tal fine, scrivete una procedura `main` di test.

Esercizio 6

Le funzioni `rifletti`*,* `nega` *e* `speculare`*.*

Implementate una funzione `rifletti`, di prototipo appropriato, che rifletta il contenuto della matrice `mat` attorno alla diagonale principale. Per esempio, la matrice quadrata seguente:

```
- + -
- + +
- + +
```

dopo la riflessione diviene:

```
+ + -
+ + +
- - -
```

Implementate inoltre una funzione `nega`, di prototipo appropriato, che modifichi la matrice `mat` nella matrice che ha in posizione (i, j) il carattere `+` se, e solo se, la matrice `mat` ha in posizione (i, j) il carattere `-`.

Chiamiamo *speculare* una matrice quadrata che sia uguale alla sua riflessione, per come definita sopra. Implementate una funzione `speculare`, di prototipo appropriato, che permetta di stabilire se la matrice `mat` sia speculare. Testate le vostre funzioni prima di proseguire.

Esercizio 7

La funzione `main`*.*

Scrivete un programma che svolga alcune operazioni sulla variabili globale `mat` degli esercizi precedenti, secondo le specifiche che seguono.

All'avvio, il programma inizializza la matrice con valori `+` e `-` distribuiti in modo casuale. Per generare un intero pseudo-casuale in C, usate il frammento di codice:

```
#include <time.h>
#include <stdlib.h>

srand(time(NULL));
int c = rand();
```

La funzione `srand(time(NULL))` va chiamata una sola volta in tutto il programma, per impostare il valore iniziale del generatore di numeri pseudo-casuali usato da `rand()`. Il valore intero `c` restituito da `rand` è un intero pseudo-casuale non negativo. L'espressione `c%2` valuterà dunque ad un intero pseudo-casuale compreso in $\{0, 1\}$.

Creata la matrice, il programma presenta all'utente il menu seguente:

```
1. Mostra matrice
2. Shift
3. Riflessione
4. Negazione
5. Controlla se speculare
6. Esci
```

Nel caso 1, il programma visualizza la matrice; nel caso 2, applica la funzione `shift` alla matrice; nel caso 3, applica la funzione `rifletti` alla matrice; nel caso 4, applica la funzione `nega` alla matrice; nel caso 5, applica la funzione `speculare` alla matrice, e comunica all'utente se la matrice sia speculare o no; nel caso 6, il programma termina.

Un esercizio aggiuntivo che dovrete svolgere, eventualmente al di fuori delle ore di laboratorio, consiste nel riscrivere il programma che avete appena sviluppato in questa seconda parte della lezione di modo che la variabile `mat` sia locale a `main`, e non più globale. Ciò implica che le varie funzioni che compongono il programma, per esempio `shift`, dovranno ricevere argomenti appropriati che permettano loro di leggere e modificare la variabile `mat` di `main`. Consultate il materiale delle lezioni frontali che tratta degli array multidimensionali come parametri alle funzioni. Rendete infine parametrico il numero n di righe e colonne della matrice, dando la possibilità di specificarlo da riga di comando all'avvio del programma.

Dipartimento di Matematica *Federigo Enriques*, Università degli Studi di Milano, via Cesare Saldini, 50, I-20133 Milano