LABORATORIO DI PROGRAMMAZIONE
CORSO DI LAUREA IN SICUREZZA INFORMAICA
UNIVERSITÀ DEGLI STUDI DI MILANO
2024–2025

Indice

Tratti dagli esercizi del corso del prof. Vincenzo Marra.
Ultima revisione: 15 novembre 2024.

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

Unire e dividere due array con allocazione dinamica.

Tempo: 25 min.

Si scriva un programma che legga dalla console due successione di valori `int`, inseriti dall'utente, che verranno memorizzati all'interno di due aree di memoria, allocate staticamente, di dimensione `N 100`. Chiedete all'utente per prima cosa quanti valori `int` intenda inserire per entrambi gli array. Implementate una funzione `merge` che, ricevuti tali array e il numero di elementi contenuti in essi, allochi dinamicamente un nuovo array che contenga esattamente tutti e soli i valori dei due array, e lo restituisca. Se il primo array contiene 3 valori, e il secondo ne conviene 5, l'array creato ne dovrà contenere esattamente 8.

Esercizio 5

Leggere una frase e creare una struttura dati per contenerla.

Tempo: 10 min.

Si scriva un programma che chieda all'utente quante parole voglia inserire. Tali parole (senza spazi o caratteri speciali) dovranno essere inserite in un array di puntatori a stringhe di dimensione `N = 20`. Per ciascuna di tale parole, inserite dall'utente, si allochi dinamicamente una nuova stringa della dimensione esatta della parola inserita. Stampare a schermo la frase ottenuta. Completate per farlo il codice di Fig. 1.

Parte 2. Ricorsione

Esercizio 6

Lunghezza di una stringa: implementazione ricorsiva.

Tempo: 15 min.

Scrivete una funzione di prototipo `int lung(char *)` che accetti in ingresso una stringa e ne restituisca la lunghezza. L'implementazione della funzione deve essere ricorsiva. Per convenzione, se il puntatore ricevuto dalla funzione è `NULL`, la lunghezza restituita è `-1`. Se lunghezza della stringa vuota—cioè, di un array di `char` il cui primo carattere sia `\0`—è zero. Scrivete una funzione `main` che vi permetta di testare la vostra implementazione della funzione `lung`.

Suggerimento. La lunghezza di una stringa della forma `cs`, dove `c` è un carattere ed `s` una stringa, è pari alla lunghezza di `s` più 1.

La codifica iterativa della funzione per il calcolo della lunghezza di una stringa, che avete già scritto in esercizi precedenti, è semplice come la versione ricorsiva e risulta molto più efficiente in esecuzione. L'Esercizio 6 ha quindi il solo scopo di farvi prendere dimestichezza con le implementazioni ricorsive. Ciò vale anche per altri esercizi di questa lezione.

Esercizio 7

Occorrenze di un carattere in una stringa: implementazione ricorsiva.

Tempo: 15 min.

Scrivete una funzione di prototipo `int occ(char *, char)` che accetti in ingresso una stringa e un carattere, e restituisca il numero di occorrenze del carattere nella stringa. L'implementazione della funzione deve essere ricorsiva. Per convenzione, se il puntatore ricevuto dalla funzione è `NULL`, il numero di occorrenze di qualunque carattere nella stringa è `-1`. Scrivete una funzione `main` che vi permetta di testare la vostra implementazione della funzione `occ`.

I numeri di Catalan

I *numeri di Catalan* prendono il nome dal matematico belga Eugène Charles Catalan (1814–1894). Essi si denotano con $C(n)$, dove n è un intero $\geqslant 0$, e contano il numero di cammini in una griglia quadrata di dimensione $n \times n$ che partono dall'angolo sud-ovest e arrivano all'angolo nord-est senza mai oltrepassare la diagonale sud-ovest/nord-est della griglia, impiegando solo passi verso nord ed est. Si veda la Figura 2.

I numeri di Catalan soddisfano la relazione ricorsiva:

$$C(n) = \frac{2(2n-1)C(n-1)}{n+1}$$

con condizione iniziale

$$C(0) = 1.$$

Esercizio 8

Numeri di Catalan.

Tempo: 20 min.

Scrivete un programma ricorsivo che legga in ingresso un intero $n \geqslant 0$ e calcoli e visualizzi il numero di Catalan $C(n)$. Nel caso in cui l'utente inserisca un valore che non soddisfa la condizione $n \geqslant 0$, forzate il reinserimento.

Esercizio 9

Tabulazione dei numeri di Catalan.

Tempo: 25 min.

Scrivete un programma che legga in ingresso un intero $n \geqslant 0$ e calcoli e visualizzi la sequenza $C(0), C(1), \ldots, C(n)$ di numeri di Catalan. Per esempio, nel caso in cui $n = 9$ l'output del vostro programma dovrebbe essere:

$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862.$$

Riutilizzate la funzione ricorsiva scritta per risolvere l'Esercizio 8.

I numeri di Delannoy

I *numeri di Delannoy* prendono il nome dall'ufficiale francese Henri-Auguste Delannoy (1833–1915), matematico dilettante. Essi si denotano con $D(m, n)$, dove m ed n sono interi $\geqslant 0$, e contano il numero di cammini in una griglia rettangolare di dimensione $m \times n$ che partono dall'angolo sud-ovest e arrivano all'angolo nord-est, impiegando solo passi verso nord, est e nord-ovest. Si veda la Figura 3.

I numeri di Delannoy soddisfano la relazione ricorsiva:

$$D(m, n) = D(m-1, n) + D(m, n-1) + D(m-1, n-1) \tag{1}$$

con condizioni iniziali

$$D(0, n) = D(m, 0) = 1. \tag{2}$$

Esercizio 10

Numeri di Delannoy.

Tempo: 20 min.

Scrivete un programma ricorsivo che legga in ingresso due interi $m, n \geqslant 0$ e calcoli e visualizzi il numero di Delannoy $D(m, n)$. Nel caso in cui l'utente inserisca valori che non soddisfano la condizione $m, n \geqslant 0$, forzate il reinserimento.

Esercizio 11

Tabulazione dei numeri di Delannoy.

Tempo: 25 min.

Scrivete un programma che legga in ingresso due interi $m, n \geqslant 0$ e calcoli e visualizzi una tabella rettangolare di dimensioni $m \times n$ il cui elemento di posto (i, j) sia il numero di Delannoy $D(i, j)$. Per esempio, nel caso in cui $m = n = 9$ l'output del vostro programma dovrebbe collimare con quello mostrato in Figura 4. Riutilizzate la funzione ricorsiva scritta per risolvere l'Esercizio 10.

I numeri di Bell

I *numeri di Bell* prendono il nome dallo storico della matematica statunitense di origini scozzesi Eric Temple Bell (1883–1960). Essi si denotano con $B(n)$, dove n è un intero $\geqslant 0$. Per definizione, $B(n)$ è il numero di partizioni di un insieme di cardinalità n.

I numeri di Bell soddisfano la relazione ricorsiva

$$B(n) = \sum_{k=0}^{n-1} \binom{n-1}{k} B(k) \tag{†}$$

per $n \geqslant 1$, con condizioni iniziali

$$B(0) = B(1) = 1.$$

Nella (†), $\binom{n-1}{k}$ è il *coefficiente binomiale*, per definizione pari al numero di sottoinsiemi di cardinalità k di un insieme di $n-1$ elementi, per $n \geqslant 1$. Il coefficiente binomiale soddisfa l'identità

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{*}$$

per $n \geqslant k \geqslant 0$. Nella (*), $n!$ denota il *fattoriale* di n, per definizione pari al numero delle permutazioni di un insieme di n elementi. Esso soddisfa la relazione ricorsiva

$$n! = n((n-1)!)$$

per $n \geqslant 1$, con condizione niziale

$$0! = 1.$$

Si veda la Figura 5.

Esercizio 12

Numeri di Bell.

Tempo: 25 min.

Scrivete una funzione `int bell(int n)` che, dato in ingresso un numero naturale $n \geqslant 0$, restituisca il numero di Bell $B(n)$. Per implementare `int bell(int n)` ricorsivamente, scrivete due funzioni ausiliarie. La prima, `int fatt(int n)`, restituisce il fattoriale del numero naturale in ingresso $n \geqslant 0$, ed è implementata ricorsivamente. La seconda, `int binom(n,k)`, restituisce il coefficiente binomiale $\binom{n}{k}$, dati due naturali in ingresso $n \geqslant k \geqslant 0$, ed è implementata usando `int` `fatt(int n)`. Scrivete una procedura `main` che chieda all'utente di inserire un intero $n \geqslant 0$ e visualizzi sul terminale il numero di Bell $B(n)$. Forzate il reinserimento se l'intero n digitato dall'utente non soddisfa la condizione $n \geqslant 0$.

Esercizio 13

Successioni maschio-femmina di Hofstadter.

Tempo: 30 min.

Si consideri la coppia di successioni definite da

$$F(n) = n - M(F(n-1))\,, \tag{3}$$

$$M(n) = n - F(M(n-1))\,, \tag{4}$$

con valori iniziali

$$F(0) = 1\,, \tag{5}$$

$$M(0) = 0\,. \tag{6}$$

Si tratta delle *successioni maschio-femmina di Hofstadter*. Per maggiori informazioni si veda, ad esempio, questa pagina di Wolfram's MatWorld.

I primi dieci numeri della sequenza maschio $M(n)$ sono $0, 0, 1, 2, 2, 3, 4, 4, 5, 6$. I primi dieci numeri della sequenza femmina $F(n)$ sono $1, 1, 2, 2, 3, 3, 4, 5, 5, 6$.

Si implementi, secondo le seguenti specifiche, un programma per calcolare i primi $n+1$ numeri delle sequenze $F(n)$ e $M(n)$, per un valore n inserito dall'utente.

(a) Il programma chiede all'utente di inserire un numero intero $n \geqslant 0$, verificando la correttezza dell'input. Se $n < 0$ il programma forza il reinserimento.

(b) Il programma visualizza in output i primi $n+1$ numeri delle sequenze $F(n)$ e $M(n)$. Ad esempio, per $n = 4$ l'output del programma sarà:

```
n   F(n)  M(n)
0    1     0
1    1     0
2    2     1
3    2     2
4    3     2
```

Dipartimento di Informatica, Università degli Studi di Milano,

stringheMalloc.c

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

#define N 20
#define MAX_WORD_LENGTH 50

int main() {
    // Chiede all'utente quanti parole vuole inserire
    int numParole, i;
    // Alloca staticamente un array di puntatori a stringhe di dimensione N
    char *parole[N];
    // Buffer per la lettura di ogni parola
    char buffer[MAX_WORD_LENGTH];

    printf("Inserisci il numero di parole (massimo %d): ", N);
    scanf("%d", &numParole);

    // Verifica che il numero di parole sia valido
    if (numParole <= 0 || numParole > N) {
        printf("Numero non valido. Assicurati di inserire un numero compreso tra 1 e %d.\n", N);
        return 1; // Termina il programma con un codice di errore
    }

    // Pulisce il buffer di input
    while (getchar() != '\n');

    // Loop per l'inserimento delle parole
    for (i = 0; i < numParole; ++i) {
        // Chiede all'utente di inserire una parola
        printf("Inserisci la parola %d: ", i + 1);
        scanf("%s", buffer);
        // Alloca dinamicamente una nuova stringa della dimensione esatta della parola
        INSERITE QUI LA VOSTRA ISTRUZIONE
        // Copia la parola nel nuovo spazio allocato
        strcpy(parole[i], buffer);
    }

    // Stampa la frase ottenuta
    printf("\nFrase ottenuta: ");
    for (i = 0; i < numParole; ++i) {
        printf("%s ", parole[i]);
    }
    printf("\n");

    // Libera la memoria allocata dinamicamente
    for (i = 0; i < numParole; ++i) {
        free(parole[i]);
    }

    return 0; // Termina il programma con successo
}
```

Figura 1. Codice parziale della soluzione.

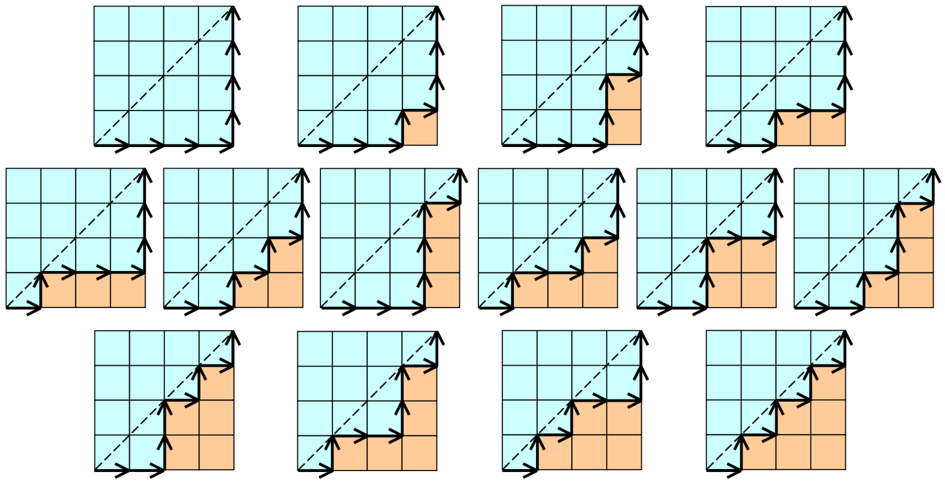

FIGURA 2. I 14 cammini di Catalan sulla griglia 4×4.

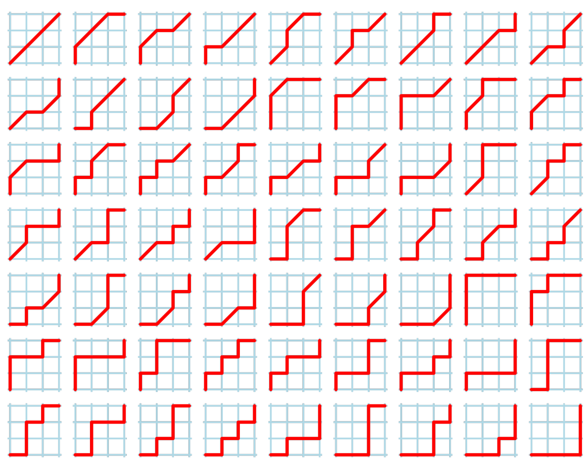

FIGURA 3. I 63 cammini di Delannoy sulla griglia 3×3.

1	1	1	1	1	1	1	1	1	1
1	3	5	7	9	11	13	15	17	19
1	5	13	25	41	61	85	113	145	181
1	7	25	63	129	231	377	575	833	1159
1	9	41	129	321	681	1289	2241	3649	5641
1	11	61	231	681	1683	3653	7183	13073	22363
1	13	85	377	1289	3653	8989	19825	40081	75517
1	15	113	575	2241	7183	19825	48639	108545	224143
1	17	145	833	3649	13073	40081	108545	265729	598417
1	19	181	1159	5641	22363	75517	224143	598417	1462563

FIGURA 4. I numeri di Delannoy $D(i,j)$ con $i, j \in \{0, 1, \ldots, 9\}$.

n	0	1	2	3	4	5	6	7	8	9
$B(n)$	1	1	2	5	15	52	203	877	4140	21147

n	0	1	2	3	4	5	6	7	8	9
$n!$	1	1	2	6	24	120	720	5040	40320	362880

n	$\binom{n}{0}$	$\binom{n}{1}$	$\binom{n}{2}$	$\binom{n}{3}$	$\binom{n}{4}$	$\binom{n}{5}$	$\binom{n}{6}$	$\binom{n}{7}$
0	1							
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
5	1	5	10	10	5	1		
6	1	6	15	20	15	6	1	
7	1	7	21	35	35	21	7	1

FIGURA 5. Tabulazione dei numeri di Bell, del fattoriale e del coefficiente binomiale.