Eserciziario di Programmazione

Roberto Borelli

Aprile 2023

Indice

Prefazione

Always remember, however, that there's usually a simpler and better way to do something than the first way that pops into your head. Donald Knuth

Sono raccolti in questo documento una serie di semplici esercizi di programmazione utili per la preparazione in vista di un compito o esame. I primi esercizi di ogni sezione servono per introdurre gradualmente l'argomento e dovrebbero risultare abbastanza facili. Se si ha difficoltà anche nei primi esercizi di ogni capitolo, allora è necessario rivedere e ristudiare i concetti. In rete sono reperibili parecchie guide, e qualche referenza la si può trovare sul mio sito `robertoborelli.com/teaching`. Gli esercizi marcati con $\star$ richiedono idee più elaborate e possono essere saltati dal lettore meno esperto. Nel capitolo 8 sono presentate delle possibili solouzioni ad una selezione degli esercizi proposti. Nell'appendice A viene proposta una breve introduzione alla ricorsione che può essere utile al fine dello svelgimento degli esercizi del capitolo 4.

Il linguaggio di programmazione. Gli esercizi sono pensati per essere indipendenti dal linguaggio di progrmmazione e dunque possono essere svolti con il vostro linguaggio preferito (es. C, C++, Java, Python,...). Le soluzioni che propongo sono scritte in C o C++ che sono i linguaggi che solitamente vengono insegnati negli istituti tecnici.

Pubblico. Il documento è rivolto principalmente a due categorie di lettori.

- **Ai tutor:** Ho utilizzato in prima persona questo eserciziario con gli studenti a cui ho dato ripetizioni negli scorsi 3/4 anni. Solitamente questo materiale va bene per preparare gli studenti della classe terza di un ITT in indirizzo informatico. Ho usato questo eserciziario anche con studenti del primo anno di università delle facoltà scientifiche come quelle di ingegneria. Per questi ultimi studenti è tuttavia necessario affrontare un numero di argomenti molto più vasto e soprattutto sarebbero necessari esercizi più articolati e complessi. Per gli studenti universitari di informatica questo documento non è per nulla adatto e dovreste cercare qualcos'altro che vada più a fondo, tuttavia potreste iniziare dagli esercizi $\star$.
- **Agli studenti:** Potete utilizzare questo eserciziario per fare palestra sui primissimi concetti di programmazione. Tenete presente che questi esercizi non coprono tutti gli argomenti di base e probabilmente online si possono trovare eserciziari migliori che contengono spiegazioni più dettagliate e precise.

Reperibilità. La versione più aggiornata del documento è reperibile all'inidirzzo `robertoborelli.com/teaching`. Se il link non dovesse più funzionare segnalatemelo.

Segnalazioni. Eventuali errori, modifiche e aggiunte possono essere segnalate scrivendomi una mail a `borelli.roberto@spes.uniud.it` o contattandomi in altro modo. Si ringrazia fin da ora chi contribuirà a mantenere l'eserciziario aggiornato.

Capitolo 1

Esecuzione condizionale

Alcuni esercizi di base che necessitano solo della conoscenza degli operatori matematici, dell'uso delle variabili e dei costrutti if. Un'ottimo punto di partenza.

Iniziamo!

Esercizio 1.1 (Moltiplicazione)**.** L'utente inserisce un numero x. Stampare $5x$ (il numero inserito dall'utente moltiplicato per 5).

Esercizio 1.2 (Divisione)**.** L'utente inserisce due numeri, stampare la loro divisione. Attenzione: un numero non può essere diviso per 0.

Esercizio 1.3 (Controllo parità)**.** L'utente inserisce un numero intero x. Stampare *pari* se x è pari, stampare *dispari* altrimenti.

Esercizio 1.4 (13-divisibilità)**.** L'utente inserisce n. Calcolare se n è divisibile per 13 e stampare il risultato.

Esercizio 1.5 (y-divisibilità)**.** L'utente inserisce due numeri x e y, stampare se x è divisibile per y.

Esercizio 1.6 (Elevazione a potenza)**.** L'utente inserisce due numeri x e y, stampare x^y.

Esercizio 1.7 (Arrotondamento)**.** (Soluzione a pagina 34)
L'utente inserisce un numero decimale nella forma $x.y$, il programma stampa $x + 1$ se $y >= 0.5$, stampa x se $y < 0.5$.
Esempi di eseczuione:
Input: 52.51 *Output*: 53
Input: 48.13 *Output*: 18
Input: 4 *Output*: 4
Input: 5.5 *Output*: 6

Esercizio 1.8 (Convertitore ore)**.** Creare un programma che converta le ore in secondi. Viene chiesto all'utente di inserire il numero di ore e il programma stampa il risultato della conversione in secondi.
Esempi di eseczuione:
Input: 1 *Output*: 3600
Input: 1.2 *Output*: 4320
Input: 2 *Output*: 7200

Esercizio 1.9 (Equazioni di primo grado)**.** Un'equazione di primo grado è del tipo $ax + b = 0$. L'utente inserisce a e b, stampare la soluzione dell'equazione oppure un messaggio di errore qualora i dati inseriti non ne permettano il calcolo.
Esempi di eseczuione:
Input: $a = 2$, $b = 3$ *Output*: -1.5
Input: $a = 2$, $b = 0$ *Output*: 0
Input: $a = 0$, $b = 2$ *Output*: Non esiste soluzione.

Esercizio 1.10 (Grandezza numero)**.** L'utente inserisce un numero n. Stampare *"grande"* se $n > 1000$, stampare *"medio"* se $n <= 1000$ e $n > 100$, stampare *"piccolo"* se $n <= 100$.
Esempi di eseczuione:
Input: 2 *Output*: *"piccolo"*
Input: 1122 *Output*: *"grande"*

Esercizio 1.11 (Equazioni di secondo grado)**.** (Soluzione a pagina 35)
Un'equazione di secondo grado è del tipo $ax^2 + bx + c = 0$. L'utente inserisce a, b c, stampare le due soluzioni dell'equazione qualora entrambe esistano, stampare l'unica soluzione dell'equazione se $a = 0$ oppure un messaggio di errore se i dati inseriti non permettono il calcolo di nessuana delle soluzioni.
Esempi di eseczuione:
Input: $a = 0$, $b = 2$, $c = 3$ *Output*: Esiste solo una soluzione e vale -1.5
Input: $a = 1$, $b = 2$, $c = 1$ *Output*: Esiste solo una soluzione e vale -1
Input: $a = 0$, $b = 2$, $c = 0$ *Output*: Esiste solo una soluzione e vale 0
Input: $a = 2$, $b = 2$, $c = 2$ *Output*: Non esiste soluzione.
Input: $a = 2$, $b = 2$, $c =$ *-4 Output*: *x1 = -2, x2 = 1*
Input: $a = 1$, $b = 2.2$, $c = 1$ *Output*: *x1 = -0.64, x2 = -1.56*

Esercizio 1.12 (Pitagora)**.** L'utente inserisce tre numeri a, b, c. Creare un programma che stampi *yes* se i tre numeri rappresentano una terna pitagorica, e che stampi *no* altrimenti.
Esempi di eseczuione:
Input: 3 4 5 *Output*: yes
Input: 5 4 3 *Output*: yes
Input: 3 3 3 *Output*: no
Input: 12 5 13 *Output*: yes

Valutazione di codice con esecuzione condizionale

Esercizio 1.13 (Valutazione flusso di codice 1)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
if (5 < 3){
  cout << 1;
}else{
  cout << 2;
}
cout << 3;
```

Esercizio 1.14 (Valutazione flusso di codice 2)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
int y = 10;
int z = 30;
if (y == 3){
```

```
  cout << 1;
}else if (z == 30){
  cout << 2;
}else{
  cout << 3;
}

int x = z;
if (y == 10 && x == 30){
  cout << 4;
}
```

Esercizio 1.15 (Valutazione flusso di codice 3)**.** Dire cosa stampa il seguente pezzo di codice C++ nel caso in cui inzialmente d valga 1 e nel caso in cui valga 17.

```
int d;
if (d < 10){
  d++;
}else if  (d < 19){
  d = d + 2;
}

if (d < 20){
  d = d + 4;
}

cout << d;
```

Esercizio 1.16 (Valutazione flusso di codice 4)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
int a = 0, b = 2, c = 1;
bool d = true, e = false;
char f = 'a', g = 'b';

if(a == 1 || ( d && g == 'b')){
  e = !d;
  b++;
  c--;
  a = b + c + a + a;
}else{
  a++;
}

if(e){
  a--; b--; c--;
  f = g;
}else if(2 < 3 && !e){
  f = g;
  g = f;
}else{
  b++;
}

a = a + b + c;

cout << a << b << c;
cout << d << e;
cout << f << g;
```

Capitolo 2

Cicli e Array

Esercizi base su cicli e array

Alcuni semplici esercizi da svolgersi con procedure iterative (cicli for, while, do while...) tramite l'ausilio della struttura dati *array* qualora sia necessario o richiesto.

Esercizio 2.1 (Stampa carattere)**.** L'utente inserisce un carattere c e un numero n. Il programma stampa in output c (il carattere inserito) per n volte.
Esempi di eseczuione:
Input: c = 'h', n = 3 *Output*: hhh
Input: c = 'z', n = 8 *Output*: zzzzzzzz

Esercizio 2.2 (Stampa numeri natuali)**.** L'utente inserisce un numero n. Il programma stampa i numeri compresi tra 0 e n.
Esempi di eseczuione:
Input: 1 *Output*: 0 1
Input: 7 *Output*: 0 1 2 3 4 5 6 7

Esercizio 2.3 (Stampa numeri pari)**.** L'utente inserisce un numero n. Il programma stampa i numeri pari compresi tra 0 e n.
Esempi di eseczuione:
Input: 1 *Output*: 0
Input: 2 *Output*: 0 2
Input: 7 *Output*: 0 2 4 6
Input: 8 *Output*: 0 2 4 6 8

Esercizio 2.4 (Stampa array)**.** Creare un array di interi di dimensione 8 che contenga i seguenti elementi: 3, 5, 78, 98, 43, 97, 09, 1.

A. Stampare a video gli elementi contenuti nell'array.

B (Stampa inversa 1)**.** Stampare a video gli elementi contenuti nell'array in ordine inverso.

Esercizio 2.5 (Somma 10 numeri)**.** L'utente inserisce 10 numeri. Il programma ne calcola la somma.
Esempi di eseczuione:
Input: 1 2 3 4 5 6 7 8 9 10 *Output*: 55
Input: 1 1 1 1 1 1 1 1 1 1 *Output*: 10

Esercizio 2.6 (Somma ad elementi array)**.** Creare un array di interi di dimensione 7 che contenga gli elementi come in tabella 2.1.

Tabella 2.1: Disposizione iniziale elementi array.

2	6	9	10	12	5	7

A. Creare un programma che sommi 5 a tutti gli elementi contentuti nell'array. Ossia a fine computazione, l'array deve contenere gli elementi come in tabella 2.2.

Tabella 2.2: Disposizione finale elementi array.

7	11	14	15	17	10	12

B. Creare un programma che chieda all'utente di inserire un numero n. Il programma deve sommare n a tutti gli elementi contenuti nell'array.
Esempi di eseczuione:
Input: n = 1 *Output*:

3	7	10	11	13	6	8

Input: n = -2 *Output*:

0	4	7	8	10	3	5

Esercizio 2.7 (Somma elementi array)**.** (Soluzione a pagina 36)
Dato un array e la sua lunghezza, calcolare la somma di tutti gli elementi presenti in esso.
Esempi di eseczuione:
Input:

5	9	12	13	15	8	10

Output: 72
Input:

9	9

Output: 18

Esercizio 2.8 (Media array)**.** Dato un array di interi, creare un programma che calcoli la media dei suoi elementi.
Esempi di eseczuione:
Input:

1	0	3	2	3	0	5

Output: 2

Esercizio 2.9 (Somma di array)**.** Creare un programma che prenda in input due array a e b entrambi di dimensione n. Creare un array c in cui ogni cella di c contenga la somma delle crispettive celle di a e b nella medesima poszione.

Esempi di eseczuione:
Input:

5	9	12

,

1	2	3

Output:

6	11	15

Input:

3	4

,

1	3

Output:

4	7

Esercizio 2.10 (Moda array 1)**.** (Soluzione a pagina 36)
Dato un array contenente numeri compresi tra 0 e 999, creare un programma che calcoli la moda dei suoi elementi. Se la moda non esiste, restituire un elemento qualsiasi il cui numero di occorenze è massimo.
Esempi di eseczuione:
Input:

1	0	3	2	998	0	5

Output: 0
Input:

1	1	3	1	0	0	5

Output: 1
Input:

1	1	3	1	0	0	0

Output: 0 oppure 1

Esercizio 2.11 (Riempimento array)**.** Creare un array di dimensione 10. Riempire le 10 caselle dell'array con 10 numeri inseriti dall'utente. Stampare i valori contenuti nell'array.

Variante A. Creare un array di dimensione n (dove n è un numero inserito dall'utente). Riempire le n caselle dell'array con n numeri inseriti dall'utente. Stampare i valori contenuti nell'array.

Esercizio 2.12 (Stampa inversa 2)**.** Stampare 5 numeri inseriti dall'utente in ordine inverso.

Variante A. Stampare n numeri inseriti dall'utente in ordine inverso, dove n è anch'esso un numero inserito dall'utente.

Esercizio 2.13 (Numeri casuali)**.** Creare un array di dimensione 10. Inserire nelle caselle dell'array 10 numeri generati casualmente. Stampare i valori contenuti nell'array.

Variante A. Creare un array di dimensione n (dove n è un numero inserito dall'utente). Inserire nelle caselle dell'array n numeri generati casualmente. Stampare i valori contenuti nell'array.

Esercizio 2.14 (Massimo e minimo)**.** Dato un array di interi inserito dall'utente, calcolare e stampare il massimo numero presente nell'array.
Esempi di eseczuione:
Input: | 5 | 9 | 12 | 13 | 15 | 8 | 10 | *Output*: 15
Input: | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 | *Output*: 523

Variante A. Dato un array di interi inserito dall'utente, calcolare e stampare il minimo numero presente nell'array.

Variante B. Dato un array di interi e un numero a, calcolare e stampare il minimo numero presente nell'array che risulta essere maggiore maggiore di a.

Esempi di eseczuione:
Input: | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, 11 *Output*: 12
Input: | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, 20 *Output*: 50

Esercizio 2.15 (Appartenenza)**.** Dato un array un numero x, determinare se x compare nell'array.

Esempi di eseczuione:
Input: | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, $x = 2$ *Output*: no
Input: | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, $x = 13$ *Output*: yes
Input: | 3 | 9 | 12 | 12 | 12 | 8 | 10 | 50 |, $x = 12$ *Output*: yes

Esercizio 2.16 (Inclusione)**.** Dato un array un array a e un array b, determinare se b compare interamente nell'array a in qualche posizione. In altre parole, se n_b rappresenta la lunghezza di b e n_a la lunghezza di a, determinare se esiste una posizione i (dove $0 \leq i \leq n_a - n_b$) tale che $a[i] = b[0]$ e $a[i+1] = b[1]$ e ... e $a[i + n_b - 1] = b[n_b - 1]$.
Esempi di eseczuione:
Input: a : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, b : | 3 | 9 | 13 | *Output*: no
Input: a : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, b : | 15 | 8 | 10 | 50 | *Output*: yes
Input: a : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, b : | 15 | 8 | 10 | 50 | 3 | *Output*: no
Input: a : | 3 | 9 | 12 |, b : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 | *Output*: no

Esercizio 2.17 (Inclusione circolare)**.** Dato un array a (lungo n_a) e un array b (lungo n_b), determinare se b compare interamente nell'array a in qualche posizione. L'array a è circolare, ossia per ogni numero $i \geq 0$ possiamo accedere alla posizione $a[i]$ dove $a[i]$ rappresenta l'elemento di a in posizione $n_a\%i$. Chiediamo quindi di determinare se esiste una posizione i (dove $0 \leq i \leq n_a - 1$) tale che $a[i] = b[0]$ e $a[i+1] = b[1]$ e ... e $a[i + n_b - 1] = b[n_b - 1]$.

Esempi di eseczuione:

Input: a : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, b : | 3 | 9 | 13 | *Output*: no

Input: a : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, b : | 15 | 8 | 10 | 50 | *Output*: yes

Input: a : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, b : | 15 | 8 | 10 | 50 | 3 | *Output*: yes

Input: a : | 3 | 9 | 12 |, b : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 | *Output*: no

Esercizio 2.18 (Slice)**.** Dato un array a (lungo n) e due numeri i e j entrambi non negativi e entrambi minori di n tali che $i \leq j$, stampare lo slice di a tra le posizioni i e j. In altre parole stampare gli elementi $a[i]$, $a[i+1]$, $a[i+2]$, ..., $a[j]$.

Esempi di eseczuione:

Input: a : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, i = 2, j = 3 *Output*: 12, 13

Input: a : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |, i = 3, j = 3 *Output*: 13

Input: a : | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 |,i = 3, j = 7 *Output*: 13, 15, 8, 10, 50

Esercizio 2.19 (Elementi ripetuti 1)**.** Dato un array, determinare *se esiste* un elemento che compare *almeno due* volte nell'array.

Esempi di eseczuione:

Input: | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 | *Output*: no

Input: | 3 | 9 | 12 | 12 | 15 | 8 | 10 | 50 | *Output*: yes

Input: | 3 | 9 | 12 | 12 | 12 | 8 | 10 | 50 | *Output*: yes

Esercizio 2.20 (Elementi ripetuti 2)**.** Dato un array, determinare *se esiste* un elemento che compare *almeno tre* volte nell'array.

Esempi di eseczuione:

Input: | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 | *Output*: no

Input: | 3 | 9 | 12 | 12 | 15 | 8 | 10 | 50 | *Output*: no

Input: | 3 | 9 | 12 | 12 | 12 | 8 | 10 | 50 | *Output*: yes

Esercizio 2.21 (Elementi ripetuti 3)**.** Dato un array, determinare *se ogni* elemento dell'array compare *esattamente* due volte.

Esempi di eseczuione:

Input: | 3 | 9 | 12 | 13 | 15 | 8 | 10 | 50 | *Output*: no

Input: | 3 | 9 | 12 | 12 | 15 | 8 | 10 | 50 | *Output*: no

Input: | 3 | 3 | 12 | 12 | 8 | 8 | 10 | 10 | *Output*: yes

Esercizio 2.22 ($\star$ Select)**.** Dati un array a e un numero intero i, calcolare l'elemento che finirebbe in posizione i, se a fosse ordinato in maniera crescente.

Esercizio 2.23 (Controllo intervallare 1)**.** L'utente inserisce una sequenza di 10 numeri interi e in seguito due numeri a e b.

A. Calcolare e stampare quanti numeri della sequenza sono maggiori di a.

B. Calcolare e stampare quanti numeri della sequenza sono minori di b.

C. Calcolare e stampare quanti numeri della sequenza sono contemporaneamente maggiori di a e minori di b.

Esempi di eseczuione:
Input: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; $a = 2$, $b = 5$ *Output*: 8 numeri sono maggiori di a, 4 numeri sono minori di b, 2 numeri sono maggiori di a e minori di b

Esercizio 2.24 (Controllo intervallare 2)**.** L'utente inserisce una sequenza di 10 numeri interi e in seguito due numeri a e b entrambi maggiori di 0.

- Se nella sequenza è stato inserito almeno un numero maggiore di a, ma nessun numero è maggiore di b, stampare la somma dei 10 numeri.
- Se nella sequenza è stato inserito almeno un numero maggiore di a, e almeno un numero è maggiore di b, stampare la media dei 10 numeri.
- Se nella sequenza tutti i numeri sono minori di a ma $b <= 10$, stampare $\sqrt{a^2 + b^4 + \sqrt{b}}$
- Altrimenti stampare in ordine inverso tutti i 10 numeri.

Esempi di eseczuione:
Input: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; $a = 2$, $b = 50$ *Output*: 55
Input: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; $a = 2$, $b = 2$ *Output*: $5, 5$
Input: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; $a = 50$, $b = 2$ *Output*: $50, 173839\ldots$
Input: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; $a = 50$, $b = 15$ *Output*: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,

Esercizio 2.25 (Somma di Gauss)**.** Creare un programma che prenda in input dall'utente un numero intero n e stampi il risultato della somma dei numeri compresi tra 0 e n.
Esempi di eseczuione:
Input: 4 *Output*: 10 *(spiegazione:* $0 + 1 + 2 + 3 + 4 = 10$*.)*
Input: 5 *Output*: 15
Input: 240 *Output*: 28920

Variante A. Creare un programma che prenda in input dall'utente un numero intero n e stampi il risultato della somma dei numeri *pari* compresi tra 0 e n.
Esempi di eseczuione:
Input: 4 *Output*: 6 *(spiegazione:* $0 + 2 + 4 = 6$*.)*

Variante B. Creare un programma che prenda in input dall'utente un numero intero n e stampi il risultato della somma dei *quadrati* dei numeri compresi tra 0 e n.
Esempi di eseczuione:
Input: 4 *Output*: 30 *(spiegazione:* $0^2 + 1^2 + 2^2 + 3^2 + 4^2 = 30$*.)*

Esercizio 2.26 (Radice quadra n volte)**.** Creare un programma che prenda in input un numero x e un numero intero n e stampi in output la radice quadrata applicata n volte ad x.
Esempi di eseczuione:

Input: 256 1 *Output*: 16 *(spiegazione:* $\sqrt{256} = 16$*.)*
Input: 256 3 *Output*: 2 *(spiegazione:* $\sqrt{\sqrt{\sqrt{256}}} = 2$*.)*
Input: 43046721 4 *Output*: 3

Esercizio 2.27 (Tabelline)**.** Creare un programma che prenda in input due numeri x e y e che stampi in output la tabellina delle moltiplicazioni del numero x a partire da 1 fino a y.
Esempi di eseczuione:
Input: 3 4 *Output*: 3x1=3 3x2=6 3x3=9 3x4=12
Input: 9 5 *Output*: 9x1=9 9x2=18 9x3=27 9x4=36 9x4=45

Variante A. Creare un programma che prenda in input tre numeri x, y, z e che stampi in output la tabellina delle moltiplicazioni del numero x a partire da y fino a z.

Esempi di eseczuione:
Input: 10 31 34 *Output*: 10x31=310 10x32=320 10x33=330 10x34=340

Esercizio 2.28 (Shifting)**.** (Soluzione a pagina 37)
Sia a un array di dimensione 5 contenenga gli elementi come in tabella 2.3.

Tabella 2.3: Disposizione iniziale elementi array.

4	5	8	10	12

A. Creare un programma che sposti (*shifti*) tutti gli elementi dell'array di una posizione a sinistra. Il valore della prima cella verrà invece spostato nell' ultima. Ovvero a fine computazione, l'array deve contenere gli elementi come in tabella 2.4.

Tabella 2.4: Disposizione finale elementi array.

5	8	10	12	4

B. Creare un programma analaogo al quello del punto A ma che sposti gli elementi a destra. A fine fine computazione, l'array deve contenere gli elementi come in tabella 2.5.

Tabella 2.5: Disposizione finale elementi array.

12	4	5	8	10

C. Modificare il programma della richiesta A tale che:

- Prenda in input n numeri inseriti dall'utente e li memorizzi nelle prime n celle di un opportuno array.
- Esegua l'operazione di *shifting* a sinistra di una posizione come nel punto A.
- Stampi a video tutte le celle dell'array a computazione avvenuta.

Esercizio 2.29 (Majority candidate)**.** In Nepal viene svolto un referendum popolare a due risposte: *sì* e *no*. I voti degli N cittadini vengono memorizzati in un array *votiCittadini* che in ogni cella contiene un numero tra $-1, 0, 1$. In particolare un -1 indica che un cittadino si è rifiutato di votare. Uno 0 indica che un cittadino ha votato no, e un 1 indica che un cittadino ha votato sì.

Tabella 2.6: Esempio di riempimento: In Nepal ci sono 10 cittadini: 6 si sono rifiutati di votare, 3 hanno votato no, e 1 ha votato sì.

votiCittadini	-1	0	-1	-1	-1	-1	1	0	0	-1

A. Creare una funzione *contaOccorrenze(arr, x)*, che dato un array *arr* e un numero x, conti il numero di occorrenze di x all'interno di *arr*. Ad esempio considerando l'array *votiCittadini* della tabella 2.6,
Esempi di eseczuione:
Input: contaOccorrenze(votiCittadini, -1) *Output*: 6
Input: contaOccorrenze(votiCittadini, 0) *Output*: 3
Input: contaOccorrenze(votiCittadini, 1) *Output*: 1
Input: contaOccorrenze(votiCittadini, 62) *Output*: 0

B. Sfruttando la funzione *contaOccorrenze(arr, x)* della richiesta A, creare una funzione *quorumRaggiunto(votiCittadini)*, che dato l'array *votiCittadini*, restituisca *true* se e soltanto se più del 50% dei cittadini ha votato, ossia se nell'array il numero di -1 è minore o uguale al 50% del totale delle celle. Nella tabella 2.6 il quorum non è raggiunto.

C. Sfruttando la funzione *contaOccorrenze(arr, x)* della richiesta A, e la funzione *quorumRaggiunto(votiCittadini)* della richiesta B, creare un programma che dato l'array *votiCittadini* inserito dall'utente dia in output uno dei seguenti esiti:

- Quorum non raggiunto.
- Quorum raggiunto, riforma approvata.
- Quorum raggiunto, riforma non approvata.

Matrici

Esercizio 2.30 (Matrice identità)**.** Creare una funzione che prenda in input un numero n e allochi e restituisca la matrice identità $n \times n$.
Esempi di eseczuione:
Input: 3 *Output*: $\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$

Esercizio 2.31 (Matrice nulla)**.** Creare una funzione che prenda in input un numero n e allochi e restituisca la matrice $n \times n$ composta da tutti 0.
Esempi di eseczuione:
Input: 3 *Output*: $\left(\begin{smallmatrix}0&0&0\\0&0&0\\0&0&0\end{smallmatrix}\right)$

Esercizio 2.32 (Traccia)**.** Creare una funzione che prenda in input una matrice $n \times n$ di numeri interi e restituisca la sua traccia.
Esempi di eseczuione:
Input: $\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$ *Output*: 3
Input: $\left(\begin{smallmatrix}1&0&0\\0&2&0\\0&0&3\end{smallmatrix}\right)$ *Output*: 6

Esercizio 2.33 (Da matrice a colonna)**.** Creare una funzione che prenda in input una matrice $m \times n$ (m righe, n colonne) di numeri interi e un numero i, e allochi e restituisca un array (lungo m) contenente l'i-ma colonna della matrice.

Esempi di eseczuione:
Input: $\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$, $i = 2$ *Output*: $\begin{array}{|c|c|c|}\hline 0&0&1\\\hline\end{array}$
Input: $\left(\begin{smallmatrix}1&5&0\\0&2&0\\0&8&3\end{smallmatrix}\right)$, $i = 1$ *Output*: $\begin{array}{|c|c|c|}\hline 5&2&8\\\hline\end{array}$

Esercizio 2.34 (Matrice trasposta)**.** Creare una funzione che prenda in input una matrice $m \times n$ (m righe, n colonne) e allochi e restituisca la matrice trasposta $n \times m$ (n righe, m colonne). *Suggerimento*: Usare la funzione creata per l'esercizio 2.33.
Esempi di eseczuione:
Input: $\left(\begin{smallmatrix}1&2&3\\4&5&6\end{smallmatrix}\right)$ *Output*: $\left(\begin{smallmatrix}1&4\\2&5\\3&6\end{smallmatrix}\right)$
Input: $\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$ *Output*: $\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$

Esercizio 2.35 (Da matrice a riga)**.** Creare una funzione che prenda in input una matrice $m \times n$ (m righe, n colonne) di numeri interi e un numero i, e allochi e restituisca un array (lungo n) contenente l'i-ma riga della matrice. Per l'implementazione non si proceda come fatto nell'esercizio 2.33 ma si sfruttino invece le funzioni create per gli esercizi 2.33 e 2.34. *Suggerimento*: Se A Ã¨ una matrice, a cosa corrisponde una riga di A nella matrice trasposta A^T?
Esempi di eseczuione:
Input: $\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$, $i = 2$ *Output*: $\begin{array}{|c|c|c|}\hline 0&0&1\\\hline\end{array}$
Input: $\left(\begin{smallmatrix}1&5&0\\0&2&0\\0&8&3\end{smallmatrix}\right)$, $i = 1$ *Output*: $\begin{array}{|c|c|c|}\hline 0&2&0\\\hline\end{array}$

Esercizio 2.36 (Moltiplicazione per uno scalare)**.** Creare una funzione che prenda in input una matrice $m \times n$ (m righe, n colonne) di numeri interi e un numero λ e calcoli la matrice che risulta moltiplicando per λ ogni elemento della matrice.
Esempi di eseczuione:
Input: $\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$, $\lambda = 2$ *Output*: $\left(\begin{smallmatrix}2&0&0\\0&2&0\\0&0&2\end{smallmatrix}\right)$
Input: $\left(\begin{smallmatrix}1&5&0\\0&2&0\end{smallmatrix}\right)$, $\lambda = 5$ *Output*: $\left(\begin{smallmatrix}5&15&0\\0&10&0\end{smallmatrix}\right)$

Esercizio 2.37 (Somma di tutte le righe)**.** Creare una funzione che prenda in input una matrice $m \times n$ (m righe, n colonne) di numeri interi e un numero j e allochi e restituisca un array lungo m in cui in ogni posizione i troviamo la somma di tutti gli elementi della matrice nella riga i.
Esempi di eseczuione:
Input: $\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$, $j = 2$ *Output*: $\begin{array}{|c|c|c|}\hline 1&1&1\\\hline\end{array}$
Input: $\left(\begin{smallmatrix}1&5&0\\0&2&0\end{smallmatrix}\right)$, $j = 5$ *Output*: $\begin{array}{|c|c|}\hline 6&2\\\hline\end{array}$

Esercizio 2.38 (Somma di tutti gli elementi)**.** Creare una funzione che prenda in input una matrice $m \times n$ (m righe, n colonne) di numeri interi e allochi e restituisca un numero che rappresenta la somma di tutti gli elementi. *Suggerimento*: Per l'implementazione usare la funzione dell'esercizio 2.37 e poi ...
Esempi di eseczuione:
Input: $\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right)$ *Output*: 3
Input: $\left(\begin{smallmatrix}1&5&0\\0&2&0\end{smallmatrix}\right)$ *Output*: 8

Esercizio 2.39 (Aggiunta di una riga)**.** Creare una funzione che prenda in input una matrice A di dimensioni $m \times n$ (m righe, n colonne) di numeri interi e un array b di n elementi. La funzione deve

allocare e restituire la matrice $(m+1) \times n$ che si ottiene aggiungendo in fondo alla matrice A una riga contenente i valori del vettore b.
Esempi di eseczuione:
Input: $A = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$, $b = \boxed{1}\boxed{2}\boxed{3}$ *Output*: $\begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 3 \end{pmatrix}$
Input: $A = \begin{pmatrix} 1 & 5 & 0 \\ 0 & 2 & 0 \end{pmatrix}$, $b = \boxed{0}\boxed{2}\boxed{3}$ *Output*: $\begin{pmatrix} 1 & 5 & 0 \\ 0 & 2 & 0 \\ 0 & 2 & 3 \end{pmatrix}$

Numeri primi e rappresentazione dei numeri

Esercizio 2.40 (Test primalità)**.** Dato in input un numero n inserito dall'utente, calcolare con una procedura $isPrime(n)$ se n è primo o meno.
Esempi di eseczuione:
Input: 5 *Output*: primo
Input: 6 *Output*: composto
Input: 9 *Output*: composto

Esercizio 2.41 (Tutti i numeri primi)**.** Dato in input un numero n inserito dall'utente, stampare con una procedura $printAllPrimes(n)$ tutti i numeri primi più piccoli di n.

Esercizio 2.42 (Rappresentazione binaria)**.** Dato in input un numero decimale n inserito dall'utente, calcolare la rappresentazione binaria di n.
Esempi di eseczuione:
Input: 3 *Output*: 11
Input: 8 *Output*: 1000

Esercizio 2.43 (Rappresentazione decimale)**.** Dato in input un numero binario n inserito dall'utente, calcolare la rappresentazione decimale di n.
Esempi di eseczuione:
Input: 100 *Output*: 4

Esercizio 2.44 (Convertitore da qualsiasi base a qualsiasi base 1)**.** Creare un convertitore di numeri interi da qualsiasi base a qualisasi base. L'utente inserisce $base_{partenza}$ e $base_{arrivo}$, viene poi inserito un numero x rappresentato in $base_{partenza}$, stampare x in $base_{arrivo}$.

Esercizio 2.45 (Convertitore da qualsiasi base a qualsiasi base 2)**.** Creare un convertitore come l'esercizio 2.44 che funzioni anche sui numeri con la virgola.

Esercizio 2.46 ($\star$ Convertitore da qualsiasi base a qualsiasi base 3)**.** Possiamo rappresentare un numero periodico con il proprio periodo preposto da un carattere p. Ad esempio $0.\overline{3}$ può essere scritto come $0.p3$. $12.57\overline{82}$ può essere scritto come $12.57p82$. Estendere il convertitore dell'esercizio 2.45 ai numeri periodici. Attenzione: un numero NON periodico in base b_1 può invece esserlo in b_2. Il programma deve essere quindi in grado di ricevere in input numeri periodici con la notazione sopra descritta e, se necessario, fornire in output numeri periodici.

Stampa di asterischi

Esercizio 2.47 (Stampa asterischi 1)**.** Dato un numero x inserito dall'utente, il programma stampa un quadrato formato da asterischi, avente lato x.
Esempi di eseczuione:
Input: $x = 3$ *Output*:

***
***
***

Input: $x = 1$ *Output*:
*

Input: $x = 2$ *Output*:
**
**

Esercizio 2.48 (Stampa asterischi 2)**.** Dato un numero x inserito dall'utente, il programma stampa un triangolo formato da asterischi, avente base e altezza pari a x.

Esempi di eseczuione:
Input: $x = 2$ *Output*:
*
**

Input: $x = 4$ *Output*:
*
**
***
****

Valutazione di codice iterativo

Esercizio 2.49 (Valutazione flusso di codice 5)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
for(int i = 0; i < 10; i = i+2){
  cout << i << endl;}
```

Esercizio 2.50 (Valutazione flusso di codice 6)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
for(int i = 1; i <= 13; i = i*2){
  cout << i << endl;}
```

Esercizio 2.51 (Valutazione flusso di codice 7)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
for(int i = 1; i <= 13; i--){
  cout << i << endl;}
```

Esercizio 2.52 (Valutazione flusso di codice 8)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
for(int i = 5; i > -3; i = i - 3){
  cout << i << endl;}
```

Esercizio 2.53 (Valutazione flusso di codice 9)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
for(int i = 1; i <= 10; i++){
  cout << (i*2)-1 << endl;}
```

Esercizio 2.54 (Valutazione flusso di codice 10)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
for(int i = 1; i <= 10; i++){
  if(i % 3 == 1){
    cout << (i*2)-1 << endl;
  }else{
    cout << i << endl;
  }}
```

Esercizio 2.55 (Valutazione flusso di codice 11)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
for(int i = 0; i < 10; i++){
  if(i % i == 0){
    cout << i+1 << endl;
  }}
```

Esercizio 2.56 (Valutazione flusso di codice 12)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
int arr[4] = {1,5,3,4};
for(int i = 0; i < 4; i++){
  cout << arr[i] + i << endl;}
```

Esercizio 2.57 (Valutazione flusso di codice 13)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
int arr[4] = {1,2,3,4};
for(int i = 0; i < 3; i++){
  cout << arr[i+1] << endl;}
cout << arr[2];
```

Esercizio 2.58 (Valutazione flusso di codice 14)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
int arr[4] = {1,2,3,4};
for(int i = 1; i < 4; i++){
  arr[i] = arr[i-1] * 2;}
cout << arr[2];
```

Esercizio 2.59 (Valutazione flusso di codice 15)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
int arr[4] = {4,3,3,2};
for(int i = 1; i < 4; i++){
  if(arr[i] % 2 == 0){
    cout << arr[i-1] << endl;
  }}
```

Esercizio 2.60 (Valutazione flusso di codice 16)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
int arr[8] = {4,3,3,2,4,8,5,9};
for(int i = 1; i < 7; i++){
  if(arr[i] % 2 == 0){
    arr[i] = arr[i-1] + arr[i];
  }}
cout << arr[5];
```

Esercizio 2.61 (Valutazione flusso di codice 17)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
for(int i = 1; i <= 10; i++){
  for(int j = 10; j > 0; j--){
    cout << j;
  }}
```

Esercizio 2.62 (Valutazione flusso di codice 18)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
for(int i = 5; i < 10; i++){
  for(int j = 5; j >= 0; j--){
    cout << i;
  }}
```

Esercizio 2.63 (Valutazione flusso di codice 19)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
for(int i = 0; i <= 10; i++){
  for(int j = 0; j <= i; j++){
    cout << i;
  }}
```

Capitolo 3

Stringhe

Alcuni esercizi sulle stringhe di testo. Se necessario usari cicli e array.

Esercizi di base con le stringhe e caratteri

Esercizio 3.1 (Lunghezza)**.** Data una parola in input, stampare la sua lunghezza.
Esempi di eseczuione:
Input: "ciao" *Output*: 4
Input: "arnaldo" *Output*: 7

Esercizio 3.2 (Vocali)**.** Dato un carattere inserito dall'utente, stampare *sì* se è una vocale, *no* altrimenti.
Esempi di eseczuione:
Input: "a" *Output*: sì
Input: "r" *Output*: no

Esercizio 3.3 (Conta vocali)**.** Data una parola inserita dall'utente, stampare il numero di vocali che essa contiene.
Esempi di eseczuione:
Input: "ciao" *Output*: 3
Input: "ronaldo" *Output*: 3
Input: "friggitrice" *Output*: 4
Input: "https" *Output*: 0

Esercizio 3.4 (Decisione di un numero binario)**.** Data una stringa s stampare in output "sì" se s rappresenta un numero binario "no" altrimenti.
Esempi di eseczuione:
Input: 0 *Output*: sì
Input: 1 *Output*: sì
Input: 1003 *Output*: no
Input: 239 *Output*: no
Input: 1011 *Output*: sì

Esercizio 3.5 (Decisione di un numero in base k)**.** Data una stringa s e un numero k con $k >= 2$ stampare in output "sì" se s rappresenta un numero in base k "no" altrimenti.

Suggerimento: Come è fatto un numero in base k? Si noti che questo problema Ã¨ solo una generalizzazione dell'esercizio 3.4.

Esercizio 3.6 (Conta caratteri)**.** Data una parola in input.

A. Contare (e stampare in output) quanti caratteri della parola sono 'a'.
Esempi di eseczuione:
Input: "ciao" *Output*: 1
Input: "arnaldo" *Output*: 2
Input: "Arnaldo" *Output*: 1 *(spiegazione: viene conteggiata solo la 'a' minuscola.)*
Input: "Roberto" *Output*: 0

B. Dato un carattere c e una parola p in input, contare quante volte compare c in p.
Esempi di eseczuione:
Input: a "arnaldo" *Output*: 2
Input: r "arnaldo" *Output*: 1
Input: m "arnaldo" *Output*: 0

Esercizio 3.7 (Convertitore a maiuscole)**.** Data una parola in input (che può contenere sia lettere minuscole che maiuscole), stampare in output la stessa parola ma con tutti i caratteri in maiuscolo.

Esempi di eseczuione:
Input: "ciao" *Output*: CIAO
Input: "ciAo" *Output*: CIAO

Esercizio 3.8 (Incremento binario)**.** Data una stringa s che rappresenta un numero binario num, stampare in output il sucessore di num.
Esempi di eseczuione:
Input: 0 *Output*: 1
Input: 1 *Output*: 10
Input: 100 *Output*: 101
Input: 111 *Output*: 1000
Input: 1011 *Output*: 1100

Esercizio 3.9 (Parola palindroma)**.** (Soluzione a pagina 38)
Data una parola in input dire se è una parola palindroma.
Esempi di eseczuione:
Input: "ciao" *Output*: no
Input: "ciaooaic" *Output*: sì
Input: "ciaoaic" *Output*: sì
Input: "2002" *Output*: sì
Input: "2003" *Output*: no

Esercizio 3.10 (Da stringa a numeri)**.** Dato un numero in input, stampare una stringa che contenga le cifre del numero ≪ in lettere ≫.
Esempi di eseczuione:
Input: "1" *Output*: uno
Input: "11" *Output*: uno, uno
Input: "1234" *Output*: uno, due, tre, quattro

Input: "2002" *Output*: due, zero, zero, due

Esercizio 3.11 (Stringa inversa)**.** Data una stringa in input, stampare la sua inversa.
Esempi di eseczuione:
Input: "1" *Output*: "1"
Input: "ciao" *Output*: "oaic"
Input: "come stai ?" *Output*: "? iats emoc"

Esercizio 3.12 (Plurale inglese)**.** Data una parola singolare in lingua inglese, stampare il plurale.
Esempi di eseczuione:
Input: "dog" *Output*: "dogs"
Input: "hater" *Output*: "haters"
Input: "banana" *Output*: "bananas"

Esercizio 3.13 (Tempo infinito)**.** Dato un verbo (in italiano) al tempo infinito, stampare lo stesso verbo al tempo participio passato.
Esempi di eseczuione:
Input: "parlare" *Output*: "parlato"
Input: "subire" *Output*: "subito"
Input: "premere" *Output*: "premuto"

Esercizio 3.14 (Riconosci coniugazione)**.** Dato un verbo (in italiano) al tempo infinito, stampare 1, 2 o 3 rispettivamente se il verbo è della prima, seconda o terza coniugazione.
Esempi di eseczuione:
Input: "arrangiare" *Output*: 1
Input: "mentire" *Output*: 3
Input: "vedere" *Output*: 2

Esercizio 3.15 (Complemento a uno)**.** Data una sequenza di bit, stampare il complemento a uno.

Esempi di eseczuione:
Input: 11111 *Output*: 00000
Input: 101 *Output*: 010
Input: 11110000111010101 *Output*: 00001111000101010

String Matching

Esercizio 3.16 (String Matching Esatto (1))**.** Siano date due stringhe, T (il testo) e P (il pattern). Stampare *la prima* posizione in cui P compare in T.
Esempi di eseczuione:
Input: T=aaaa, P=a *Output*: 0
Input: T=bbbaaaa, P=a *Output*: 3
Input: T=bbbabaaaa, P=ba *Output*: 2
Input: T=bbbabaaaa, P=bab *Output*: 2
Input: T=bbbabaaaa, P=baba *Output*: 2
Input: T=bbbabaaaa, P=babab *Output*: error
Input: T=aaaa, P=b *Output*: error

Esercizio 3.17 (String Matching Esatto 2)**.** Siano date due stringhe, T (il testo) e P (il pattern). Stampare *tutte* le posizioni in cui P compare in T.
Esempi di eseczuione:
Input: T=aaaa, P=a *Output*: 0,1,2,3
Input: T=bbbaaaa, P=a *Output*: 3,4,5,6
Input: T=bbbabaaaa, P=ba *Output*: 2,4
Input: T=bbbabaaaa, P=bab *Output*: 2
Input: T=bbbabaaaa, P=baba *Output*: 2
Input: T=bbbabaaaa, P=babab *Output*: error
Input: T=aaaa, P=b *Output*: error

Esercizio 3.18 ($\star$ String Matching Esatto 3)**.** Considerare l'esercizio 3.18, sia n la lunghezza di P, sia m la lunghezza di T. Possiamo costruire un algoritmo che risolve l'esercizio in tempo $O(mn)$? possiamo farlo in $O(m+n)$? possiamo farlo in meno tempo?

Capitolo 4

Ricorsione

Alcuni classici esercizi da svolgere con procedure ricorsive che non facciano uso di cicli. Per esercizio provare anche a svolgerli con procedure iterative e riflettere sulle differenze implementative.
Per una piccola introduzione all'argomento vedere l'appendice A a pag. 43

Esercizi di base sulla ricorsione

Esercizio 4.1 (Stampa inversa)**.** Caricare 5 numeri in un array e stamparli in ordine inverso con una procedura ricorsiva.

Esercizio 4.2 (Fattoriale)**.** (Spiegazione a pag. 46) Calcolare il numero $n!$ con una procedura ricorsiva.

Esercizio 4.3 (Parola palindroma)**.** Data una stringa di testo s, creare una procedura $isPalindrome(s)$ che restiruisca true se e soltanto se s è palindroma.

Esercizio 4.4 (Fibonacci 1)**.** Creare una procedura ricorsiva $fibRec(n)$ che calcoli l'ennesimo numero di fibonacci. Si rciorda che i numeri di fibonacci sono definiti come segue: $f(0) = 1, f(1) = 1$ e $f(n) = f(n-1) + f(n-2)$ se $n \geq 2$.

Esercizio 4.5 ($\star$ Fibonacci 2)**.** Creare una procedura ricorsiva $fibRecSmart(i)$ che operi come $fibRec(n)$ dell'esercizio 4.4 ma che abbia una complessità polinomiale sull'input.

Esercizio 4.6 (Massimo)**.** Calcolare il massimo numero presente in un array di interi con una procedura ricorsiva.

Esercizio 4.7 (Moltiplicazione)**.** (Soluzione a pagina 39)
Calcolare la moltiplicazione di due numeri x e y con una procedura ricorsiva. *Suggerimento*: ad ogni passo sommare x alla variabile $risultato$ e decrementare y di 1.

Esercizio 4.8 (Riempimento array)**.** Dato un array a di dimensione n, inizializzare tutti i suoi valori a x (dove x è inserito dall'utente) con una procedura ricorsiva.

Esercizio 4.9 (Somma array)**.** Calcolare la somma di tutti gli elementi presenti in un array con una procedura ricorsiva.

Allineamento DNA: Longest common subsequence

Una sequenza di DNA è una successione di caratteri appartenenti all'alfabeto {A, G, C, T}. Un'allineamento è una procedura con cui vengono messe a confronto ed allineate due o più sequenze primarie di aminoacidi, DNA o RNA. L'allineamento permette di individuare regioni identiche o simili. Un'allineamento di due stringhe consiste, in pratica, nel trovare una sottoseqeunza di caratteri

comune (CS: common subsequence). Date due stringhe possiamo elencare tutte le CS di lunghezza 2, analogamente possiamo elencare tutte le CS di lunghezza 3, tutte le CS di lunghezza 4 e così via. Date due stringhe possiamo affermare che esiste una lunghezza massima per le sottosequenze comuni (LLCS: length of longest common subsequence). Data la LLCS invece possono esistere più sottosequenze con la stessa lunghezza.
Ad esempio le sequenze di dna AGACTGAACATAC e GATCCGACTAC ammettono varie sottosequenze comuni (ad esempio AG che ha lunghezza 2). La lunghezza massima degli allineamenti è 9, tuttavia con questa lunghezza possiamo trovare due CS GACGACTAC e GATGACTAC. Le tabelle 4.1 e 4.2 mostrano effettivamente come questi allineamenti siano trovati. (I caratteri in rosso non fanno parte del CS e sono quindi scartati).

Tabella 4.1: Allineamento GACGACTAC

A	G	A		C	T	G	A	A	C	A	T	A	C
	G	A	T	C	C	G	A		C		T	A	C

Tabella 4.2: Allineamento GATGACTAC

A	G	A	C	T		G	A	A	C	A	T	A	C
	G	A		T	C	G	G	A	C		T	A	C

Passando ad un esempio più semplice, analizzando le parole PANE e CANE, gli allineamenti di lunghezza 1 sono: {A, N, E}, gli allineamenti di lunghezza 2 sono {CA, NE, AE} e l'allineamento (unico in questo caso) di lunghezza massima (3) è { ANE }

Tabella 4.3: Allineamento ANE

P	A	N	E
C	A	N	E

Esercizio 4.10 (⋆ Lcs 1)**.** (Soluzione a pagina 39)
Creare una procedura $llcs(x, y)$ che prenda in input due stringhe x e y e determini la lunghezza massima che può avere un allineamento tra le due stringhe.

Esercizio 4.11 (⋆ Lcs 2)**.** Creare una procedura $lcs(x, y)$ che prenda in input due stringhe x e y e determini UNO tra gli allineamenti di x e y a lunghezza massima.

Esercizio 4.12 (⋆ Lcs 3)**.** Creare una procedura $lcsMem(x, y)$ che proceda come $lcs(x, y)$ dell'esercizio 4.11 ma che abbia complessità polinimiale rispetto all'input applicando una delle tecniche di programmazione dinamica.

Esercizio 4.13 (⋆ Lcs 4)**.** Creare una procedura $allLcs(x, y)$ che prenda in input due stringhe x e y e determini TUTTI gli allineamenti di x e y a lunghezza massima.

Valutazione di codice ricorsivo

Esercizio 4.14 (Valutazione flusso di codice 20)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
f(int x){
  if(x < 10){
    return f(x+5);
  }
  else{
    retrun x;
  }
```

```
}
int main(){
  cout << f(2) << f(40) << f(3);
}
```

Esercizio 4.15 (Valutazione flusso di codice 21)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
f(int x){
  if(x < 10){
    return f(f(x+10));
  }
  else{
    retrun x;
  }
}

int main(){
    cout << f(1) + f(2) + f(3);
}
```

Esercizio 4.16 (Valutazione flusso di codice 22)**.** Dire cosa stampa il seguente pezzo di codice C++.

```
f(int x, int y){
  if(x < 5){
    return f(y + 2, x);
  }
  else{
    retrun y;
  }
}

int main(){
    cout << f(2,2) * f(3,3);
}
```

Capitolo 5

Liste concatenate

Questo capitolo è solo una bozza preliminare. Consideriamo le liste concatenate come una struttura in cui ogni nodo ha due campi `value` e `next`.

Esercizio 5.1 (Ribalta)**.** Data una lista l, creare una lista con gli stessi elementi di l ma che appaiono in ordine inverso.
Esempi di eseczuione:
Input: `[1] -> [2] -> [6] -> [3]` *Output*: `[3] -> [6] -> [2] -> [1]`

Capitolo 6

Ordinamento

Alcuni problemi riguardanti l'ordinamento di un array di interi.

Esercizio 6.1 (Masimo e minimo di vettore ordinato)**.** (Soluzione a pagina 41)
Creare due procedure $orderedMax(a)$ e $orderedMin(a)$ che prendano in input un array ordinato in modo crescente e calcolino in modo efficente, rispettivamente il massimo e il minimo dell'array.

Esercizio 6.2 (Ricerca)**.** Creare una procedura efficente $orderedSearch(a, k)$ che dati un'array a e un numero k, restituisca l'indice i tale che $a[i] = k$ se tale i esiste. In altri termini la procedura deve restituire la posizione di k nell'array a, se k appartiene ad a.

Variante A. Creare una procedura $orderedSearchRec(a, k)$ che operi in maniera ricorsiva.

Esercizio 6.3 (Scambia)**.** Creare una procedura $swap(a, i, j)$ che prenda in input un array a e due indici i e j e scambi i valori contenuti nelle due posizioni $a[i]$ e $a[j]$ se i due inidici sono validi.

Esercizio 6.4 (Inserimento ordinato 1)**.** Creare una procedura iterativa $orderedInsert(a, i, x)$ che prenda in input un array a ordinato dalla posizione 0 alla poszione i estremi compresi, e un numero intero x. Se a contiene più di $i+1$ celle, x viene inserito in a nella posizione posizione corretta ossia tale che dopo l'inserimento, a dalla posizione 0 alla poszione $i+1$ sia un array ordinato. In altri termini, x viene inserito in una posizione j tale che:

$$a[0] \leq \cdots \leq a[j-1] \leq a[j] \leq a[j+1] \leq \cdots \leq a[i+1]$$

Se lo ritieni opportuno utilizza le procedure *merge* e *swap* dei punti 6.7 e 6.3.

Esercizio 6.5 (Inserimento ordinato 2)**.** Data la procedura $orderedInsert$ dell'esercizio 6.4, creare la procedura $orderedInsertRec$ che si comporta allo stesso modo ma opera in maniera ricorsiva.

Esercizio 6.6 (InsertionSort)**.** Implementare $insertionSort$.

Esercizio 6.7 (Merge)**.** Creare una procedura $merge(a, b)$ che prenda in input due array ordinati a e b e restitruisca un array ordinato che contiene tutti e soli gli elementi di a e b. Se lo ritieni opportuno utilizza la procedura *swap* del punto 6.3.

Esercizio 6.8 (MergeSort)**.** Implementare $mergeSort$.

Esercizio 6.9 (k-sum)**.** Dato un array ordinato, e un numero k, determinare se nell'array sono presenti due numeri x ed y tali che $x + y = k$. Nel caso esistano, restituire x ed y. Nota: x ed y devono trovarsi in due posizioni diverse dell'array.

Esercizio 6.10 (Moda array 2)**.** (Soluzione a pagina 41)
Dato un array ordinato, determinare con una procedura lineare la moda dell' array.

Esercizio 6.11 (Elementi ripetuti 4)**.** (Soluzione a pagina 41)
Dato un array *ordinato* a, determinare se ci sono ripetizioni. In altre parole, determinare se esistono i e j tali che $a[i] = a[j]$.

Capitolo 7

Esercitazioni

Alcuni esercizi più lunghi e complessi che richiedono tutti o molti dei concetti visti nei capitoli precedenti.

Esercizio 7.1 (Registro palestra)**.** Si consideri una palestra in cui per ogni iscritto sono registrati:

- codice fiscale
- nome
- cognome
- peso
- altezza

Si chiede di realizzare un programma di gestione degli iscritti con le seguenti specifiche.

- Il programma inizialmente stampa il menù:
  ————————————————————————-
  Digita un numero per scegliere un operazione.
  1) Inserisci atleti.
  2) Ricerca un atleta.
  ————————————————————————-

- Se viene premuto 1, il programma consente di inserire i dati di 10 atleti.
  ————————————————————————-
  inserire atleta 1:
  RSSMRA97M25A757C mario rossi 80 180

  inserire atleta 2:
  DREVRZ85M25A769D andrea verza 90 100
  ...
  ————————————————————————-

- Se viene premuto 2, il programma chiede di inserire un codice fiscale. Il programma stamperà *non trovato* se non è stato registrato alcun iscritto con quel dato codice fiscale, altrimnenti verranno stampati tutti i suoi dati.
  ————————————————————————-

inserire un codice fiscale da ricercare:
RSSMRA97M25A757C

Atleta trovato:
Nome: Mario
Cognome: Rossi
Peso: 80
Altezza: 180
————————————————————————-

- Dopo che è stata eseguita una delle due operazioni precederenti, il programma ristampa il menù.

Per la realizzazione del programma è consigliato (ma non obbligatorio) partire dal seguente pezzo di codice:

```
string codiciFiscali[10];
string nomi[10];
string cognomi[10];
int pesi[10];
int altezze[10];

/*Funzione che stampera il menu
e chiedera all'utente di inserire un numero*/
void stampaMenu(){}

/*Funzione che chiedera all'utente di inserire i dati
di 10 atleti e li memorizzera negli appositi array*/
void inserisciAtleti(){}

/*Funzione che prende in input un codice fiscale
e stampa i dati dell'atleta corrispondente (se il codice
fiscale e' memorizzato)*/
void ricercaAtleta(string codiceFiscale){}

int main(){
        stampaMenu();
}
```

Esercizio 7.2 (Registro scolastico)**.** La segreteria scolastica di una scuola necessita di un programma che funga da registro e che consenta di gestire gli studenti di una determinata classe. Per ogni studente sono memorizzati: nome, cognome, numero di telefono, data di nascita. Il programma deve permettere di inserire un nuovo studente, modificare i dati di uno studente e visualizzare la lista di tutti gli studenti inseriti. Il programma deve avere un menù che consenta di scegliere l'operazione da svolgere:

————————————————————————-
Digita un numero per scegliere un operazione.
1) Inserisci uno studente.
2) Modifica i dati di uno studente.
3) Visualizza lista studenti.
4) Esci.
————————————————————————-

- Nel caso venga digitato 1, il programma deve chiedere nome, cognome, numero di telefono e data di nascita.

- Nel caso venga digitato 2, il programma chiede il nome dello studente di cui si vogliono modificare i dati e in seguito richiede tutti i nuovi dati.
- Nel caso venga digitato 3, il programma stampa la lista di tutti gli studenti presenti nel registro.
- Nel caso venga digitato 4, il programma termina.

Dopo che un'operazione di inserimento, modifica o visualizzazione si è conclusa, viene ristampato il menù per consentire diverse operazioni di fila. Si supponga inoltre che il registro non contenga mai più di 1000 studenti.
Ai fini della realizzazione del programma, si consiglia di utilizzare:

1. Una variabile *studentiInseriti* che tenga traccia di quanti studenti siano stati inseriti.
2. Una funzione *stampamenù* che generi il menù di cui sopra.
3. Una funzione *inserisciStudente* che prenda in input nome, cognome, telefono e data di nascita ed esegua l'operazione di inserimento nei registri.
4. Una funzione *modificaStudente* che prenda in input nome e cognome ed esegua l'operazione di modifica.
5. Una funzione *stampaListaStudenti* che stampi la lista dei studenti.

Opzionalmente, aggiungere al menù un opzione che consenta di visualizzare la lista degli studenti in ordine alfabetico.

Capitolo 8

Soluzioni

In questo capitolo sono esposte alcune soluzioni degli esercizi e alcuni suggerimenti. I codici proposti sono scritti in C salvo diversa indicazione.

Viene fortemente suggerito di **guardare le soluzioni solo se necessario** oppure solo a completamento dell'esercizio in questione. Guardare le soluzioni senza provare ad eseguire l'esercizio autonomomamente è spesso inutile e antiproduttivo. Se un esercizio non viene svolto correttamente al primo tentativo, riprovare fino a quando non si ottiene un esito positivo.

Soluzioni capitolo 1

Soluzione 1.7. L'esercizio chiede di arrotondare un numero dato, all'intero successivo o precedente in base alla parte decimale del numero stesso.
Idea:

- Leggere il numero inserito dall'utente e memorizzarlo in una variabile che possa contenere numeri con la virgola.
- Calcoliamo la parte intera inferiore del numero (ad esempio se il numero inserito è 56.8 la parte intera inferiore è 56.)
- Calcoliamo la differenza tra il numero inserito dall'utente e la parte intera inferiore.
    1. Se è zero significa che l'utente ha inserito un numero senza virgola. Stampiamo quindi il numero stesso (o la parte intera inferiore . . . sono uguali).
    2. Se è maggiore di 0 ma minore di 0.5 il numero inserito dall'utente è con la virgola ma la parte decimale è minore di 0.5. Stampiamo quindi la parte intera inferiore del numero inserito.
    3. Altrimenti il numero inserito dall'utente è con la virgola con parte decimale maggiore o uguale di 0.5. Stampiamo quindi la parte intera inferiore del numero inserito incrementata di 1.

Codice C:

```c
#include <math.h>  //libreria matematica
#include <stdio.h> //libreria per I/O

int main(){
        //leggo un numero decimale da stdin
        float x;
        scanf("%f", &x);
```

```
        //ottengo parte intera inferiore
        int parte_inferiore = floor(x);

        //calcolo la differenza e stampo output
        float diff = x - parte_inferiore;
        if(diff == 0){
                printf("%d\n", parte_inferiore);
        }else if (diff < 0.5){
                printf("%d\n", parte_inferiore);
        }else{
                printf("%d\n", parte_inferiore + 1);
        }
        return 0;
}
```

Soluzione 1.11. *Codice C:*

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

int main()
{
        double a, b, c;
        double delta;
        double x1, x2;

        printf("Inserire i valori di a, b, c: ");
        scanf("%lf %lf %lf", &a, &b, &c);

        if(a == 0){
                // l'equazione e' bx + c = 0
                if(b != 0){
                        x1 = -c/b;
                        printf("Esiste una soluzione: x=%lf \n", x1);
                }else{
                        printf("Non esistono soluzioni \n");
                }
        }else{
                delta = pow(b,2) - 4*a*c;
                if(delta == 0){
                        x1 = -b/(2*a);
                        printf("Esiste una soluzione: x=%lf \n", x1);
                }else if(delta > 0){
                        x1 = (-b + sqrt(delta)) / (2*a);
                        x2 = (-b - sqrt(delta)) / (2*a);
                        printf("x1=%lf x2=%lf \n", x1, x2);
                }else{
                        printf("Non esistono soluzioni reali \n");
                }
        }

        return 0;
}
```

Soluzioni capitolo 2

Soluzione 2.7. *Idea:*
Teniamo in memoria una variabile *somma*, scorriamo l'array e ad ogni passo incrementiamo *somma* con il valore della cella i dell'array.
Codice C:

```c
#include <stdio.h>
#define L 7
int main()
{
        int arr [L] = {5 , 9 , 12 , 13 , 15 , 8 , 10};
        int somma = 0;

        for(int i = 0; i < L; i++){
                somma = somma + arr[i];
        }

        printf("%d\n", somma);

        return 0;
}
```

Soluzione 2.10. *Idea:*
Creiamo un secondo array, *occorrenze*, e lo riempiamo tale che $occorrenze[i] = x$ significa che nell'array di input il numero i è stato trovato x volte. Ora ci basta trovare la posizione dell'array *occorrenze* contenente il numero massimo.
Codice C:

```c
#include <stdio.h>
#define N 30
int main()
{
        int array[N] = {1,90,3,4,90,6,7,90,90,34,65,78,24,
                23,999,90,23,1,2,345,345,5,23,24,25,26,27,28,29};
        int occorrenze[1000] = {};

        // costruisco array delle occorrenze
        for(int i = 0; i < N; i++){
                int el = array[i];
                occorrenze[el] ++;
        }

        // trovo l'indice del numero massimo
        // all'interno dell'array delle occorrenze
        int max = array[0];
        int indiceMax = 0;
        for(int i = 1; i < 1000; i++){
                if(occorrenze[i] > max){
                        max = occorrenze[i];
                        indiceMax = i;
                }
        }

        // La moda e' l'indice massimo
        // infatti e' l'elemento che piu occorre
        // nell array di partenza
        printf("%d\n", indiceMax);
        return 0;
}
```

Soluzione 2.28. L'eserzio chiede di shiftare a sinistra di una posizione tutte le celle di un dato array. Sia a l'array su cui staimo operando. L'array a si trova nella seguente situazione iniziale: Vogliamo ottenere:

4	5	8	10	12

5	8	10	12	4

Idea:

- Creare una variabile *temp* per contenere il valore $a[0]$.
- Considerare il numero in posizione k, $a[k]$.
- Copiare $a[k+1]$ in $a[k]$.
- Incrementare k e ripetere il procedimento.
- Se k vale 4 (cioè si sto considerando l'ultima cella disponibile dell'array), copiare *temp* in $a[4]$.

Rappresentazione grafica: *Codice C:*

1. Configurazione iniziale. | 4 | 5 | 8 | 10 | 12 |
2. Copio $a[0]$ in *temp*: | 4 |
3. Copio $a[1]$ in $a[0]$. | 5 | 5 | 8 | 10 | 12 |
4. Copio $a[2]$ in $a[1]$. | 5 | 8 | 8 | 10 | 12 |
5. Copio $a[3]$ in $a[2]$. | 5 | 8 | 10 | 10 | 12 |
6. Copio $a[4]$ in $a[3]$. | 5 | 8 | 10 | 12 | 12 |
7. Copio *temp* in $a[4]$. | 5 | 8 | 10 | 12 | 4 |
8. Configurazione finale raggiunta. | 5 | 8 | 10 | 12 | 4 |

```
int main(){
        int a [] = {4, 5, 8, 10, 12};

        int temp = a[0];
        for(int i = 0; i < 5; i++){
                a[i] = a[i+1];
        }
        a[4] = temp;

        return 0;
}
```

Il codice e l'idea sono facilmente generalizzabili ad un array di dimensione n e/o allo shifting a destra. Si lasciano come esercizio queste due varianti.

Soluzioni capitolo 3

Soluzione 3.9. L'esercizio chiede di decidere se una parola sia palindroma o meno.
Idea:

- Prendo in input una parola.
- Calcolo la sua lunghezza (che chiameremo n).
- Con un ciclo scandisco le lettere della parola.
  - Considero la lettera i, e la lettera in posizione "$n - i - 1$".
  - Se queste lettere sono diverse, per qualche valore di i, la parola non è palindroma.
  - Se alla fine del ciclo, non ho trovato un i tale che $parola[i] \neq parola[n - i - 1]$, allora la parola è palindroma.

Codice C:

```c
#include <stdio.h>  //libreria per I/O
#include <string.h> //libreria per le stringhe

int main(){
        char parola [100];   //alloco lo spazio per la parola
        scanf("%s", parola); //prendo in input una parola

        int n = strlen(parola); //calcolo la lunghezza

        for (int i = 0; i <= n/2; i++){
                if(parola[i] != parola[n-i-1]){
                        printf("parola non palindroma\n");
                        return 0;
                }
        }

        printf("parola palindroma\n");
        return 0;
}
```

Soluzioni capitolo 4

Soluzione 4.7. L'esercizio chiede di calcolare la moltiplicazione di due numeri x e y con una procedura ricorsiva.
Idea:

- Creo una procedura con due parametri: x e y.
- Se $y = 0$ allora restituisco 0.
- Altrimenti restituisco la somma di x e $x \cdot (y-1)$ dove $x \cdot (y-1)$ viene calcolato ricorsivamente.

Codice C:

```c
#include <stdio.h>

int molt_rec(int x, int y){
        if (y == 0) //caso basse
                return 0;
        else        //caso ricorsivo
                return x + molt_rec(x, y-1);
}

int main(){
        //valori di test
        printf("%d\n", molt_rec(4, 5));
        printf("%d\n", molt_rec(5, 4));
        printf("%d\n", molt_rec(6, 5));
}
```

Soluzione 4.10. *Codice C:*

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

// le posizioni valide di x vanno da 0 a xLenght-1
// le posizioni valide di y vanno da 0 a yLenght-1
// ad ogni passo considero la porzione x[xIndex,xIndex+1,...,xLength-1]
//                         e la porzione y[yIndex,yIndex+1,...,yLength-1]
// in particolare ad ogni passo considero
// i caratteri x[xIndex] e y[yIndex]

int llcs(char *x, char *y, int xIndex, int yIndex, int xLength, int yLength){
        if(xIndex >= xLength || yIndex >= yLength){

                // una delle due stringhe nelle porzioni considerate
                // e' vuota... allora le due porzioni non hanno niente in comune
                return 0;

        }else if(x[xIndex] == y[yIndex]){

                // trovo due caratteri uguali
                // li tengo in conto
                return 1 + llcs(x,y,xIndex+1,yIndex+1,xLength,yLength);

        }else{
                // ho due caratteri diversi

                // considero il carattere successivo di x
                int val1 = llcs(x,y,xIndex+1,yIndex,xLength,yLength);

                // considero il carattere successivo di y
```

```
                int val2 = llcs(x,y,xIndex,yIndex+1,xLength,yLength);

                // restituisco il massimo tra val1 e val2
                return (val1 > val2 ? val1 : val2);
        }
}

int main()
{
        char x[100];
        char y[100];
        scanf("%s %s", x, y); // chiedo in input due stringhe

        int xLength = strlen(x); // calcolo lunghezza x
        int yLength = strlen(y); // calcolo lunghezza y

        int result = llcs(x,y,0,0,xLength,yLength);

        printf("%d\n", result); // stampo il risultato
        return 0;
}
```

Soluzioni capitolo 6

Soluzione 6.1. *Idea:*
L'esercizio si può risolvere in modo banale. Essendo che l'array è ordinato, non occorre scandirlo per trovare il massimo e il minimo ma basta accedere rispettivamente all'ultima e alla prima posizione.
Codice C:

```c
// arr ordinato in maniera crescente
int orderedMax(int* arr, int l){
        return arr[l-1];
}

int orderedMin(int* arr, int l){
        return arr[0];
}
```

Soluzione 6.10. *Idea:*
L'array è ordinato, il che significa che se trovo un elemento x e dopo un certo numero di posizioni trovo un elemento diverso, y, continuando a scorrere l'array non potremo più trovare x. Gli elementi con lo stesso valore si trovano quindi in un gruppo di celle adiacenti dell'array. Per ogni gruppo di elementi, calcoliamo le occorrenze e le confrontiamo con quelle del gruppo con numero massimo di occorrenze trovato fino ad ora. Quest'idea si traduce in un unica scansione dell'array.
Codice C:

```c
// arr ordinato in maniera crescente
int orderedMode(int* arr, int l){
        int candidato;
        int occorrenze = 0;

        int occorrenzeAttuali = 1;
        int condidatoAttuale = a[0];
        for(int i = 1; i < l; i++){
                if (arr[i] == condidatoAttuale){
                        occorrenzeAttuali ++;
                }else{

                        // controllo se ho trovato un candidato migliore
                        if(occorrenzeAttuali > occorrenze){
                                candidato = condidatoAttuale;
                                occorrenze = occorrenzeAttuali;
                        }

                        // imposto il nuovo condidatoAttuale
                        condidatoAttuale = arr[i];
                        occorrenzeAttuali = 1;
                }
        }

        // eseguo il controllo per l'ultimo elemento
        if(occorrenzeAttuali > occorrenze){
                candidato = condidatoAttuale;
                occorrenze = occorrenzeAttuali;
        }

        return candidato;
}
```

Soluzione 6.11. *Idea:*
Data l'ipotesi di array ordinato, abbiamo che se esiste un elemento ripetuto allora esiste un indice i tale che $arr[i] = arr[i+1]$. In altre parole per verificare una ripetizione ci basta controllare due

celle adiacenti alla volta.
Codice C:

```c
#define TRUE 1
#define FALSE 0

// arr ordinato in maniera crescente
int elementiRipetuti (int *arr, int l){
        for(int i = 0; i < l - 1; i++){
                if(arr[i] == arr[i+1]){
                        // ho trovato un elemento ripetuto
                        return TRUE;
                }
        }
        // non ho trovato ripetizioni
        return FALSE;
}
```

Appendice A

Introduzione alla ricorsione

Definizione

Per definizione una funzione è ricorsiva se all'interno del suo corpo richiama se stessa.

```
def laMiaFunzione(...)
        ...
        laMiaFunzione(...)
        ...
```

Vediamo nelle prossime sezioni a cosa possa servire un approccio del genere e come utilizzarlo.

Scomposizione in sottoproblemi: il paradigma Divide et Impera

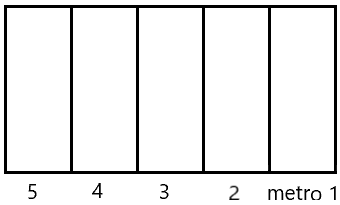
Dato un problema P, magari complesso, ci chiediamo se risolvere un sottoproblema $p1$ possa aiutarci nella soluzione del problema originale. Una volta appurato che risolvere $p1$ ci porta ad una semplice soluzione del problema P, spostiamo la nostra attenzione su $p1$. Ripetiamo il procedimento: risolvere un sottoproblema $p2$, ci aiuta a risolvere $p1$? Di fatto stiamo quindi sezionando il problema principale P, in tanti piccoli sottoproblemini $p1$, $p2$, $p3$, ... Continueremo a cercare sottoproblemi fin tanto che la soluzione di un singolo sottoproblema non sia banale e ovvia. Quindi otteniamo una sequenza finita di problemi P, $p1$, $p2$, $p3$, ..., p_n dove p_n è un problema facilmente risolvibile (caso base).

Una volta risolto il più semplice sottoproblema possiamo quindi risalire la catena e risolvere in cascata tutti i prblemi fino a risolvere P problema di partenza.

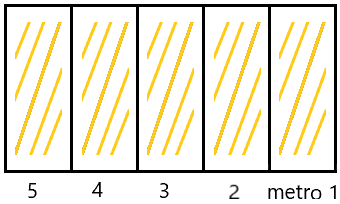
Questa tecnica di scomposizione del problema viene detta **Divide et Impera**.

Un primo esempio

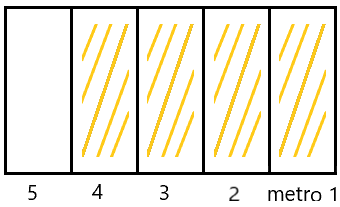
Immaginiamo una casa con una facciata lunga un certo numero di metri. Il proprietario chiama un imbianchino per pitturare di rosso i primi 5 metri della facciata.



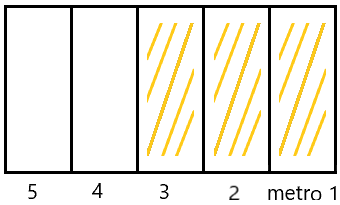
Arriva l'imbianchino: deve colorare tutti e 5 i metri:



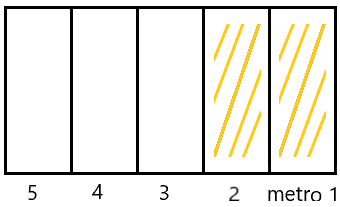
L'imbianchino dice: «per colorare 5 metri, coloro dapprima i primi 4 metri e solo dopo coloro il quinto.» L'imbianchino si concentra quindi sui primi 4 metri:



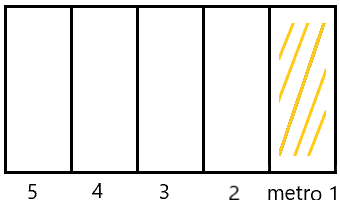
L'imbianchino dice: «per colorare 4 metri, coloro dapprima i primi 3 metri e solo dopo coloro il quarto.» L'imbianchino si concentra quindi sui primi 3 metri:



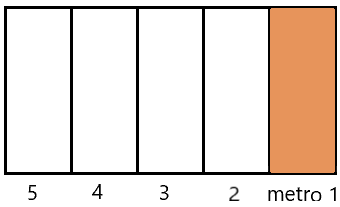
L'imbianchino dice: «per colorare 3 metri, coloro dapprima i primi 2 metri e solo dopo coloro il terzo.» L'imbianchino si concentra quindi sui primi 2 metri:



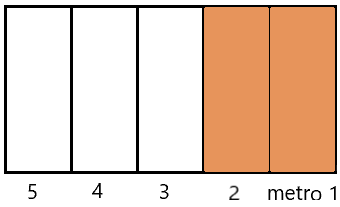
L'imbianchino dice: «per colorare 2 metri, coloro dapprima il primo metro e solo dopo coloro il secondo.» L'imbianchino si concentra quindi sul primo metro:



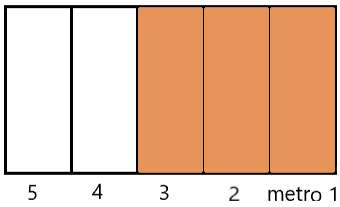
L'imbianchino si trova al primo metro e non può più rimandare il problema quindi procede colorando il primo metro.



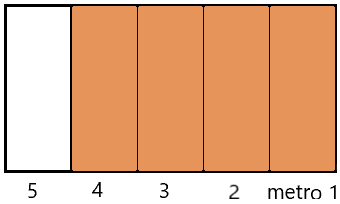
L'imbianchino ha colorato il primo metro e ora può procedere colorando il secondo come disse prima.



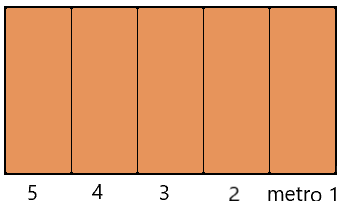
L'imbianchino ha colorato il secondo metro e ora può procedere colorando il terzo come disse prima.



L'imbianchino ha colorato il terzo metro e ora può procedere colorando il quarto come disse prima.



L'imbianchino ha colorato il quarto metro e ora può procedere colorando il quinto come disse prima.



L'imbianchino ha colorato tutti i metri e conclude il lavoro.

Quest'idea viene tradotta dal seguente codice:

```
void pitturaMuro(i){
        if(i == 1){
                pitturaMetro(1);
        }else{
                pitturaMuro(i-1);
                pitturaMetro(i);
        }
}
```

Abbiamo 2 casi per pitturare un muro di i metri:

- **Caso base:** Ho solo un metro da colorare (cioè $i == 1$) e quindi invoco $pitturaMetro(1)$, ovvero coloro questo primo metro.
- **Caso 2:** Ho più di un metro da colorare (cioè $i > 1$). Coloro dapprima i metri precedenti, e quindi invoco $pitturaMuro(i-1)$ e una volta colorati, pitturo il metro corrente $pitturaMetro(i)$.

Il calcolo fattoriale

Calcolo di $n!$
Supponendo di saper risolvere $(n-1)!$, il calcolo di $n!$ risulta facile infatti $n! = n \cdot (n-1)!$. Spostiamo quindi l'attenzione sul calcolo di $(n-1)!$, che risulterebbe facile se conoscessimo $(n-2)!$, infatti $(n-1)! = (n-1) \cdot (n-2)!$. Il sottoproblema facile in questo caso, è il calcolo di $0!$ che sappiamo risulti essere 1. Quindi:

- Vogliamo $n! \rightarrow$ ci occorre $(n-1)!$
- Vogliamo $(n-1)! \rightarrow$ ci occorre $(n-2)!$
- Vogliamo $(n-2)! \rightarrow$ ci occorre $(n-3)!$
- ...
- Vogliamo $3! \rightarrow$ ci occorre $2!$
- Vogliamo $2! \rightarrow$ ci occorre $1!$
- Vogliamo $1! \rightarrow$ ci occorre $0!$
- Conosciamo $0! = 1$
- Conosciamo $0! = 1 \rightarrow$ calcoliamo $1! = 1 \cdot 0! = 1$
- Conosciamo $1! = 1 \rightarrow$ calcoliamo $2! = 2 \cdot 1! = 2$
- Conosciamo $2! = 2 \rightarrow$ calcoliamo $3! = 3 \cdot 2! = 6$
- ...
- Conosciamo $(n-3)! \rightarrow$ calcoliamo $(n-2)! = (n-2) \cdot (n-3)!$
- Conosciamo $(n-2)! \rightarrow$ calcoliamo $(n-1)! = (n-1) \cdot (n-2)!$
- Conosciamo $(n-1)! \rightarrow$ calcoliamo $n! = n \cdot (n-1)!$
- Abbiamo risolto il problema di partenza.

Quest'idea viene tradotta dal seguente codice:

```
int fattoriale(n){
        if(n == 0){
                return 1;
        }else{
                return n * fattoriale(n-1);
        }
}
```

Distinguiamo due casi per calolare $n!$:

- **Caso base:** $n == 0$ e quindi restituisco subito 0! cioè 1.
- **Caso 2:** $n > 0$ e quindi calcolo dapprima $(n-1)!$ e poi lo moltiplico per n ottenendo quindi $n!$.

Caso base e caso ricorsivo

Come abbiamo visto nei 2 esempi precedenti, ogni funzione ricorsiva ben posta deve avere (almeno) 2 casi. Un caso base e un caso ricorsivo (anche detto passo ricorsivo o passo induttivo). Notare che nel caso ricorsivo richiamiamo la stessa funzione con un argomento « più vicino » al caso base. Se così non fosse, la funzione non terminerebbe mai. Una funzione ricorsiva è quindi del seguente tipo:

```
funzioneRicorsiva(argomento){
        if(caso_base){
                //azioni per il caso base
                //...
        }else {
                //azioni per il caso ricorsivo
                //...
                //chiamata ricorsiva
                funzioneRicorsiva(argomento - 1);
        }
}
```

Dove per «argomento - 1 » si intende un argomento più vicino al caso base.