Secondo Appello di Programmazione I

3 Febbraio 2015
Prof. Roberto Sebastiani

Codice:

Nome	Cognome	Matricola

La directory 'esame' contiene 4 sotto-directory: 'uno', 'due', 'tre' e 'quattro'. Le soluzioni vanno scritte negli spazi e nei modi indicati esercizio per esercizio.
NOTA: il codice dato non può essere modificato

Modalità di questo appello

Durante la prova gli studenti sono vincolati a seguire le regole seguenti:

- Non è consentito l'uso di alcun libro di testo o fotocopia. In caso lo studente necessitasse di carta (?), gli/le verranno forniti fogli di carta bianca su richiesta, che dovranno essere riconsegnati a fine prova. È consentito l'uso di una penna. Non è consentito l'uso di alcuno strumento calcolatore.
- È vietato lo scambio di qualsiasi informazione, orale o scritta. È vietato guardare nel terminale del vicino.
- È vietato l'uso di telefoni cellulari o di qualsiasi strumento elettronico.
- È vietato allontanarsi dall'aula durante la prova, anche se si ha già consegnato. (Ogni necessità fisiologica va espletata PRIMA dell'inizio della prova.)
- È vietato qualunque accesso, in lettura o scrittura, a file esterni alla directory di lavoro assegnata a ciascun studente. Le uniche operazioni consentite sono l'apertura, l'editing, la copia, la rimozione e la compilazione di file all'interno della propria directory di lavoro.
- Sono ovviamente vietati l'uso di email, ftp, ssh, telnet ed ogni strumento che consenta di accedere a file esterni alla directory di lavoro. Le operazioni di copia, rimozione e spostamento di file devono essere circoscritte alla directory di lavoro.
- Ogni altra attività non espressamente citata qui sopra o autorizzata dal docente è vietata.

Ogni violazione delle regole di cui sopra comporterà automaticamente l'annullamento della prova e il divieto di accesso ad un certo numero di appelli successivi, a seconda della gravità e della recidività della violazione.

NOTA IMPORTANTE: DURANTE LA PROVA PER OGNI STUDENTE VERRÀ ATTIVATO UN TRACCIATORE SOFTWARE CHE REGISTRERÀ TUTTE LE OPERAZIONI ESEGUITE (ANCHE ALL'INTERNO DELL'EDITOR!!). L'ANNULLAMENTO DELLA PROVA DI UNO STUDENTE POTRÀ AVVENIRE ANCHE IN UN SECONDO MOMENTO, SE L'ANALISI DELLE TRACCE SOFTWARE RIVELASSERO IRREGOLARITÀ.

1 Nel file `esercizio1.cc` è definito un programma per il calcolo approssimato della funzione:

$$f(x) = \frac{1}{(1-x)^2}$$

per x compreso tra -1 e $+1$, estremi esclusi, utilizzando la formula di Maclaurin. Secondo tale formula la funzione $f(x)$ può essere descritta in forma di serie come segue:

$$\frac{1}{(1-x)^2} = \sum_{i=1}^{+\infty} i x^{i-1}$$

Il programma richiede all'utente l'immissione di un numero **intero** e di un valore **double**, rappresentanti rispettivamente il numero massimo di iterazioni da compiere per calcolare l'approssimazione del valore della funzione e l'errore massimo tollerabile.

A tal proposito si assume che se $step_n$ rappresenta il valore della serie di cui sopra arrestata all'n-esimo termine, e ϵ l'errore massimo tollerabile, allora la condizione di tolleranza all'errore può essere espressa come:

$$|step_n - step_{n-1}| < \epsilon$$

Successivamente l'utente inserisce una sequenza di numeri **double**, che rappresentano i valori per cui calcolare quanto vale $f(x)$.

Si richiede di completare il programma di cui sopra inserendo la dichiarazione e la definizione della funzione "`calcola`", che, dati come parametri in ingresso

- una variabile di tipo `intero` n
- una variable di tipo `double` x
- una variable di tipo `double` *epsilon*

restituisca un valore `double` con l'approssimazione della funzione $f(x)$. Il valore *epsilon* definisce l'errore massimo tollerato; la funzione deve comunque impiegare al più n passi iterativi per completare.

È vietata ogni modifica alla funzione main pena l'annullamento dell'esercizio.

NOTA 1: È ammessa, qualora ritenuta utile al completamento dell'esercizio, la definizione di funzioni ausiliarie.

NOTA 2: Non è consentito l'uso di funzioni di libreria, né l'uso di variabili globali o di tipo `static`.

NOTA 3: Non è consentito utilizzare istruzioni `break`, `continue` o `goto`.

VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 **soluzione_A11.cc**

```
#include <iostream>
#include <iomanip>

using namespace std;

double valoreAssoluto(double valore);
double pot(double x, int n);
double calcola(double x, int n, double epsilon);

int main () {
  double epsilon, x;
  int n;

  cout << "Inserisci il numero di termini dell'approssimazione: ";
  cin >> n;
  cout << "Inserisci l'errore massimo epsilon: ";
  cin >> epsilon;

  char c;
  do {
    cout << "Inserisci il valore x in cui vuoi calcolare f(x) = 1 / (1 - x) ^ 2: ";
    cin >> x;
    while(x >= 1 || x <= -1) {
      cout << "Valore errato, prego reintrodurre (-1 < x < 1):";
      cin >> x;
    }
    cout << "L'approssimazione di f(x) e': ";
    cout << setprecision(10) << calcola(x, n, epsilon);
    cout << endl;
    cout << "Continuare (s/n)? ";
    cin >> c;
  } while(c != 'n' && c != 'N');

  return(0);
}

double valoreAssoluto(double valore) {
  return valore < 0 ? -valore : valore;
}

double pot(double x, int n) {
  // Potenza (ricorsiva)
  if (n == 0) {
    return 1.0;
  } else {
    return x * pot(x, n - 1);
  }
}

double calcola(double  x, int n, double epsilon) {
  double risultato, passo_precedente;
  // Condizioni iniziali
```

```
  risultato = 0.0;
  int i = 0;
  // Doppia condizione di uscita
  do {
    // Salvo lo step precedente
    passo_precedente = risultato;
    // Sfasatura rispetto al conteggio dei passi iterativi
    int indice = i + 1;
    // Sommo l'i-esimo termine della serie
    risultato += indice * pot(x, indice - 1);
    // Incremento il contatore
    i++;
  } while(i < n && valoreAssoluto(risultato - passo_precedente) >= epsilon);
  return risultato;
}
```

1 Nel file `esercizio1.cc` è definito un programma per il calcolo approssimato della funzione:

$$f(x) = \frac{x}{(1-x)^2}$$

per x compreso tra -1 e $+1$, estremi esclusi, utilizzando la formula di Maclaurin. Secondo tale formula la funzione $f(x)$ può essere descritta in forma di serie come segue:

$$\frac{x}{(1-x)^2} = \sum_{i=1}^{+\infty} ix^i$$

Il programma richiede all'utente l'immissione di un numero **intero** e di un valore **double**, rappresentanti rispettivamente il numero massimo di iterazioni da compiere per calcolare l'approssimazione del valore della funzione e l'errore massimo tollerabile.

A tal proposito si assume che se $step_n$ rappresenta il valore della serie di cui sopra arrestata all'n-esimo termine, e ϵ l'errore massimo tollerabile, allora la condizione di tolleranza all'errore può essere espressa come:

$$|step_n - step_{n-1}| < \epsilon$$

Successivamente l'utente inserisce una sequenza di numeri **double**, che rappresentano i valori per cui calcolare quanto vale $f(x)$.

Si richiede di completare il programma di cui sopra inserendo la dichiarazione e la definizione della funzione "`calcola`", che, dati come parametri in ingresso

- una variabile di tipo `intero` n
- una variable di tipo `double` x
- una variable di tipo `double` *epsilon*

restituisca un valore `double` con l'approssimazione della funzione $f(x)$. Il valore *epsilon* definisce l'errore massimo tollerato; la funzione deve comunque impiegare al più n passi iterativi per completare.

È vietata ogni modifica alla funzione main pena l'annullamento dell'esercizio.

NOTA 1: È ammessa, qualora ritenuta utile al completamento dell'esercizio, la definizione di funzioni ausiliarie.

NOTA 2: Non è consentito l'uso di funzioni di libreria, né l'uso di variabili globali o di tipo `static`.

NOTA 3: Non è consentito utilizzare istruzioni `break`, `continue` o `goto`.

VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 **soluzione_A12.cc**

```
#include <iostream>
#include <iomanip>

using namespace std;

double valoreAssoluto(double valore);
double pot(double x, int n);
double calcola(double x, int n, double epsilon);

int main () {
  double epsilon, x;
  int n;

  cout << "Inserisci il numero di termini dell'approssimazione: ";
  cin >> n;
  cout << "Inserisci l'errore massimo epsilon: ";
  cin >> epsilon;

  char c;
  do {
    cout << "Inserisci il valore x in cui vuoi calcolare f(x) = x / (1 - x) ^ 2: ";
    cin >> x;
    while(x >= 1 || x <= -1) {
      cout << "Valore errato, prego reintrodurre (-1 < x < 1):";
      cin >> x;
    }
    cout << "L'approssimazione di f(x) e': ";
    cout << setprecision(10) << calcola(x, n, epsilon);
    cout << endl;
    cout << "Continuare (s/n)? ";
    cin >> c;
  } while(c != 'n' && c != 'N');

  return(0);
}

double valoreAssoluto(double valore) {
  return valore < 0 ? -valore : valore;
}

double pot(double x, int n) {
  // Potenza (ricorsiva)
  if (n == 0) {
    return 1.0;
  } else {
    return x * pot(x, n - 1);
  }
}

double calcola(double  x, int n, double epsilon) {
  double risultato, passo_precedente;
  // Condizioni iniziali
```

```
  risultato = 0.0;
  int i = 0;
  // Doppia condizione di uscita
  do {
    // Salvo lo step precedente
    passo_precedente = risultato;
    // Sfasatura rispetto al conteggio dei passi iterativi
    int indice = i + 1;
    // Sommo l'i-esimo termine della serie
    risultato += indice * pot(x, indice);
    // Incremento il contatore
    i++;
  } while(i < n && valoreAssoluto(risultato - passo_precedente) >= epsilon);
  return risultato;
}
```

1 Nel file `esercizio1.cc` è definito un programma per il calcolo approssimato della funzione:

$$f(x) = \frac{2}{(1-x)^3}$$

per x compreso tra -1 e $+1$, estremi esclusi, utilizzando la formula di Maclaurin. Secondo tale formula la funzione $f(x)$ può essere descritta in forma di serie come segue:

$$\frac{2}{(1-x)^3} = \sum_{i=2}^{+\infty} (i-1)ix^{i-2}$$

Il programma richiede all'utente l'immissione di un numero **intero** e di un valore **double**, rappresentanti rispettivamente il numero massimo di iterazioni da compiere per calcolare l'approssimazione del valore della funzione e l'errore massimo tollerabile.

A tal proposito si assume che se $step_n$ rappresenta il valore della serie di cui sopra arrestata all'n-esimo termine, e ϵ l'errore massimo tollerabile, allora la condizione di tolleranza all'errore può essere espressa come:

$$|step_n - step_{n-1}| < \epsilon$$

Successivamente l'utente inserisce una sequenza di numeri **double**, che rappresentano i valori per cui calcolare quanto vale $f(x)$.

Si richiede di completare il programma di cui sopra inserendo la dichiarazione e la definizione della funzione "`calcola`", che, dati come parametri in ingresso

- una variabile di tipo `intero` n
- una variable di tipo `double` x
- una variable di tipo `double` *epsilon*

restituisca un valore `double` con l'approssimazione della funzione $f(x)$. Il valore *epsilon* definisce l'errore massimo tollerato; la funzione deve comunque impiegare al più n passi iterativi per completare.

È vietata ogni modifica alla funzione main pena l'annullamento dell'esercizio.

NOTA 1: È ammessa, qualora ritenuta utile al completamento dell'esercizio, la definizione di funzioni ausiliarie.

NOTA 2: Non è consentito l'uso di funzioni di libreria, né l'uso di variabili globali o di tipo `static`.

NOTA 3: Non è consentito utilizzare istruzioni `break`, `continue` o `goto`.

VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 **soluzione_A13.cc**

```
#include <iostream>
#include <iomanip>

using namespace std;

double valoreAssoluto(double valore);
double pot(double x, int n);
double calcola(double x, int n, double epsilon);

int main () {
  double epsilon, x;
  int n;

  cout << "Inserisci il numero di termini dell'approssimazione: ";
  cin >> n;
  cout << "Inserisci l'errore massimo epsilon: ";
  cin >> epsilon;

  char c;
  do {
    cout << "Inserisci il valore x in cui vuoi calcolare f(x) = 2 / (1 - x) ^ 3: ";
    cin >> x;
    while(x >= 1 || x <= -1) {
      cout << "Valore errato, prego reintrodurre (-1 < x < 1):";
      cin >> x;
    }
    cout << "L'approssimazione di f(x) e': ";
    cout << setprecision(10) << calcola(x, n, epsilon);
    cout << endl;
    cout << "Continuare (s/n)? ";
    cin >> c;
  } while(c != 'n' && c != 'N');

  return(0);
}

double valoreAssoluto(double valore) {
  return valore < 0 ? -valore : valore;
}

double pot(double x, int n) {
  // Potenza (ricorsiva)
  if (n == 0) {
    return 1.0;
  } else {
    return x * pot(x, n - 1);
  }
}

double calcola(double  x, int n, double epsilon) {
  double risultato, passo_precedente;
  // Condizioni iniziali
```

```
  risultato = 0.0;
  int i = 0;
  // Doppia condizione di uscita
  do {
    // Salvo lo step precedente
    passo_precedente = risultato;
    // Sfasatura rispetto al conteggio dei passi iterativi
    int indice = i + 2;
    // Sommo l'i-esimo termine della serie
    risultato += (indice - 1) * indice * pot(x, indice - 2);
    // Incremento il contatore
    i++;
  } while(i < n && valoreAssoluto(risultato - passo_precedente) >= epsilon);
  return risultato;
}
```

1 Nel file `esercizio1.cc` è definito un programma per il calcolo approssimato della funzione:

$$f(x) = \frac{2x^2}{(1-x)^3}$$

per x compreso tra -1 e $+1$, estremi esclusi, utilizzando la formula di Maclaurin. Secondo tale formula la funzione $f(x)$ può essere descritta in forma di serie come segue:

$$\frac{2x^2}{(1-x)^3} = \sum_{i=2}^{+\infty} (i-1)ix^i$$

Il programma richiede all'utente l'immissione di un numero **intero** e di un valore **double**, rappresentanti rispettivamente il numero massimo di iterazioni da compiere per calcolare l'approssimazione del valore della funzione e l'errore massimo tollerabile.

A tal proposito si assume che se $step_n$ rappresenta il valore della serie di cui sopra arrestata all'n-esimo termine, e ϵ l'errore massimo tollerabile, allora la condizione di tolleranza all'errore può essere espressa come:

$$|step_n - step_{n-1}| < \epsilon$$

Successivamente l'utente inserisce una sequenza di numeri **double**, che rappresentano i valori per cui calcolare quanto vale $f(x)$.

Si richiede di completare il programma di cui sopra inserendo la dichiarazione e la definizione della funzione "`calcola`", che, dati come parametri in ingresso

- una variabile di tipo `intero` n
- una variable di tipo `double` x
- una variable di tipo `double` *epsilon*

restituisca un valore `double` con l'approssimazione della funzione $f(x)$. Il valore *epsilon* definisce l'errore massimo tollerato; la funzione deve comunque impiegare al più n passi iterativi per completare.

È vietata ogni modifica alla funzione main pena l'annullamento dell'esercizio.

NOTA 1: È ammessa, qualora ritenuta utile al completamento dell'esercizio, la definizione di funzioni ausiliarie.

NOTA 2: Non è consentito l'uso di funzioni di libreria, né l'uso di variabili globali o di tipo `static`.

NOTA 3: Non è consentito utilizzare istruzioni `break`, `continue` o `goto`.

VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 **soluzione_A14.cc**

```
#include <iostream>
#include <iomanip>

using namespace std;

double valoreAssoluto(double valore);
double pot(double x, int n);
double calcola(double x, int n, double epsilon);

int main () {
  double epsilon, x;
  int n;

  cout << "Inserisci il numero di termini dell'approssimazione: ";
  cin >> n;
  cout << "Inserisci l'errore massimo epsilon: ";
  cin >> epsilon;

  char c;
  do {
    cout << "Inserisci il valore x in cui vuoi calcolare f(x) = 2 * x ^ 2 / (1 - x) ^ 3: ";
    cin >> x;
    while(x >= 1 || x <= -1) {
      cout << "Valore errato, prego reintrodurre (-1 < x < 1):";
      cin >> x;
    }
    cout << "L'approssimazione di f(x) e': ";
    cout << setprecision(10) << calcola(x, n, epsilon);
    cout << endl;
    cout << "Continuare (s/n)? ";
    cin >> c;
  } while(c != 'n' && c != 'N');

  return(0);
}

double valoreAssoluto(double valore) {
  return valore < 0 ? -valore : valore;
}

double pot(double x, int n) {
  // Potenza (ricorsiva)
  if (n == 0) {
    return 1.0;
  } else {
    return x * pot(x, n - 1);
  }
}

double calcola(double  x, int n, double epsilon) {
  double risultato, passo_precedente;
  // Condizioni iniziali
```

```
  risultato = 0.0;
  int i = 0;
  // Doppia condizione di uscita
  do {
    // Salvo lo step precedente
    passo_precedente = risultato;
    // Sfasatura rispetto al conteggio dei passi iterativi
    int indice = i + 2;
    // Sommo l'i-esimo termine della serie
    risultato += (indice - 1) * indice * pot(x, indice);
    // Incremento il contatore
    i++;
  } while(i < n && valoreAssoluto(risultato - passo_precedente) >= epsilon);
  return risultato;
}
```

2 Scrivere nel file `esercizio2.cc`:

(a) una funzione **ricorsiva** *calcola_norma_ricorsivo* che, dato un array di *int* e la sua dimensione, restituisca un numero *double* corrispondente alla **norma euclidea** dell'array stesso;

(b) una funzione **ricorsiva** *normalizza* che, dato un array di *int* e la sua dimensione, restituisca un **nuovo array** contenente, per ogni elemento, l'elemento corrispondente del primo array diviso per la **norma euclidea** dell'array in input, calcolata con la funzione di cui al punto precedente.

Le operazioni di calcolo della norma e di divisione degli elementi dell'array per il valore della norma vanno eseguite tramite funzioni ricorsive, **pena l'annullamento dell'esercizio**.

Non è ammesso l'uso di cicli, variabili globali o static. È ammesso l'uso di funzioni ausiliarie, purché ricorsive. È consentito l'uso della funzione `sqrt` della libreria `cmath`.

È vietata ogni modifica alla funzione main pena l'annullamento dell'esercizio.

NOTA 1: La norma euclidea di un vettore è la radice quadrata della somma dei quadrati degli elementi del vettore.

$$||x|| := \sqrt{\sum_{i=1}^{n} x_i^2}$$

Ad esempio:

- Array in input: $[1, 2, 4]$
- Norma euclidea= $\sqrt{1^2 + 2^2 + 4^2} = \sqrt{1 + 4 + 16} = \sqrt{21} = 4.58258$
- Array normalizzato = $[1/4.58258, 2/4.58258, 4/4.58258] = [0.218218, 0.436436, 0.872872]$

NOTA 2: I valori dell'array iniziale vanno inseriti utilizzando la funzione `leggi(...)` definita in `array.h` e `array.o`.
La stampa dell'array normalizzato va eseguita usando la funzione `stampa(...)` definita in `array.h` e `array.o`.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 **codice_A21.cc**

```
#include <iostream>
#include "array.h"     /* leggi, stampa, MAX_DIM */

using namespace std;

// Inserire qui sotto le DICHIARAZIONI delle funzioni

int main() {
  int array[MAX_DIM];
  int dim;
  leggi(array, dim);
  double* normalizzato = normalizza(array, dim);
  cout << "Array normalizzato: " << endl;
  stampa(normalizzato, dim);
  return 0;
}

// Inserire qui sotto le DEFINIZIONI delle funzioni
```

2 **soluzione_A21.cc**

```
#include <iostream>
#include <cmath>       /* sqrt */
#include "array.h"     /* leggi, stampa, MAX_DIM */

using namespace std;

double calcola_norma_ricorsivo(int arr[], int n, int sum);
void dividi_ricorsivo(int source[], double dest[], double norma, int n);
double* normalizza(int source[], int dim);

int main() {
  int array[MAX_DIM];
  int dim;
  leggi(array, dim);
  double* normalizzato = normalizza(array, dim);
  cout << "Array normalizzato: " << endl;
  stampa(normalizzato, dim);
  return 0;
}

double calcola_norma_ricorsivo(int arr[], int n, int sum) {
  double res;
  if (n < 0) {
    res = sqrt(sum);
  } else {
    sum += arr[n] * arr[n];
    res = calcola_norma_ricorsivo(arr, n - 1, sum);
  }
  return res;
}
```

```
void dividi_ricorsivo(int source[], double dest[], double norma, int n) {
  if (n >= 0) {
    dest[n] = source[n] / norma;
    dividi_ricorsivo(source, dest, norma, n - 1);
  }
}

double* normalizza(int source[], int dim) {
  double* normalizzato = new double[dim];
  double norma = calcola_norma_ricorsivo(source, dim - 1, 0);
  dividi_ricorsivo(source, normalizzato, norma, dim - 1);
  return normalizzato;
}
```

2 Scrivere nel file `esercizio2.cc`:

(a) una funzione **ricorsiva** *calcola_norma_ricorsivo* che, dato un array di *long* e la sua dimensione, restituisca un numero *double* corrispondente alla **norma euclidea** dell'array stesso;

(b) una funzione **ricorsiva** *normalizza* che, dato un array di *long* e la sua dimensione, restituisca un **nuovo array** contenente, per ogni elemento, l'elemento corrispondente del primo array diviso per la **norma euclidea** dell'array in input, calcolata con la funzione di cui al punto precedente.

Le operazioni di calcolo della norma e di divisione degli elementi dell'array per il valore della norma vanno eseguite tramite funzioni ricorsive, **pena l'annullamento dell'esercizio**.

Non è ammesso l'uso di cicli, variabili globali o static. È ammesso l'uso di funzioni ausiliarie, purché ricorsive. È consentito l'uso della funzione `sqrt` della libreria `cmath`.

È vietata ogni modifica alla funzione main pena l'annullamento dell'esercizio.

NOTA 1: La norma euclidea di un vettore è la radice quadrata della somma dei quadrati degli elementi del vettore.

$$||x|| := \sqrt{\sum_{i=1}^{n} x_i^2}$$

Ad esempio:

- Array in input: $[1, 2, 4]$
- Norma euclidea= $\sqrt{1^2 + 2^2 + 4^2} = \sqrt{1 + 4 + 16} = \sqrt{21} = 4.58258$
- Array normalizzato = $[1/4.58258, 2/4.58258, 4/4.58258] = [0.218218, 0.436436, 0.872872]$

NOTA 2: I valori dell'array iniziale vanno inseriti utilizzando la funzione `leggi(...)` definita in `array.h` e `array.o`.
La stampa dell'array normalizzato va eseguita usando la funzione `stampa(...)` definita in `array.h` e `array.o`.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 **codice_A22.cc**

```
#include <iostream>
#include "array.h"     /* leggi, stampa, MAX_DIM */

using namespace std;

// Inserire qui sotto le DICHIARAZIONI delle funzioni

int main() {
  long array[MAX_DIM];
  int dim;
  leggi(array, dim);
  double* normalizzato = normalizza(array, dim);
  cout << "Array normalizzato: " << endl;
  stampa(normalizzato, dim);
  return 0;
}

// Inserire qui sotto le DEFINIZIONI delle funzioni
```

2 **soluzione_A22.cc**

```
#include <iostream>
#include <cmath>       /* sqrt */
#include "array.h"     /* leggi, stampa, MAX_DIM */

using namespace std;

double calcola_norma_ricorsivo(long arr[], int n, long sum);
void dividi_ricorsivo(long source[], double dest[], double norma, int n);
double* normalizza(long source[], int dim);

int main() {
  long array[MAX_DIM];
  int dim;
  leggi(array, dim);
  double* normalizzato = normalizza(array, dim);
  cout << "Array normalizzato: " << endl;
  stampa(normalizzato, dim);
  return 0;
}

double calcola_norma_ricorsivo(long arr[], int n, long sum) {
  double res;
  if (n < 0) {
    res = sqrt(sum);
  } else {
    sum += arr[n] * arr[n];
    res = calcola_norma_ricorsivo(arr, n - 1, sum);
  }
  return res;
}
```

```
void dividi_ricorsivo(long source[], double dest[], double norma, int n) {
  if (n >= 0) {
    dest[n] = source[n] / norma;
    dividi_ricorsivo(source, dest, norma, n - 1);
  }
}

double* normalizza(long source[], int dim) {
  double* normalizzato = new double[dim];
  double norma = calcola_norma_ricorsivo(source, dim - 1, 0);
  dividi_ricorsivo(source, normalizzato, norma, dim - 1);
  return normalizzato;
}
```

2 Scrivere nel file `esercizio2.cc`:

(a) una funzione **ricorsiva** *calcola_norma_ricorsivo* che, dato un array di *int* e la sua dimensione, restituisca un numero *double* corrispondente alla **norma ad uno** dell'array stesso;

(b) una funzione **ricorsiva** *normalizza* che, dato un array di *int* e la sua dimensione, restituisca un **nuovo array** contenente, per ogni elemento, l'elemento corrispondente del primo array diviso per la **norma ad uno** dell'array in input, calcolata con la funzione di cui al punto precedente.

Le operazioni di calcolo della norma e di divisione degli elementi dell'array per il valore della norma vanno eseguite tramite funzioni ricorsive, **pena l'annullamento dell'esercizio**.

Non è ammesso l'uso di cicli, variabili globali o static. È ammesso l'uso di funzioni ausiliarie, purché ricorsive. È consentito l'uso della funzione `sqrt` della libreria `cmath`.

È vietata ogni modifica alla funzione main pena l'annullamento dell'esercizio.

NOTA 1: La norma ad uno di un vettore è la somma dei degli elementi del vettore.

$$||x|| := \sum_{i=1}^{n} x_i$$

Ad esempio:

- Array in input: $[1, 2, 4]$
- Norma $= 1 + 2 + 4 = 7$
- Array normalizzato $= [1/7, 2/7, 4/7] = [0.142857, 0.285714, 0.571428]$

NOTA 2: I valori dell'array iniziale vanno inseriti utilizzando la funzione `leggi(...)` definita in `array.h` e `array.o`.
La stampa dell'array normalizzato va eseguita usando la funzione `stampa(...)` definita in `array.h` e `array.o`.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 **codice_A23.cc**

```
#include <iostream>
#include "array.h"     /* leggi, stampa, MAX_DIM */

using namespace std;

// Inserire qui sotto le DICHIARAZIONI delle funzioni

int main() {
  int array[MAX_DIM];
  int dim;
  leggi(array, dim);
  double* normalizzato = normalizza(array, dim);
  cout << "Array normalizzato: " << endl;
  stampa(normalizzato, dim);
  return 0;
}

// Inserire qui sotto le DEFINIZIONI delle funzioni
```

2 **soluzione_A23.cc**

```
#include <iostream>
#include "array.h"     /* leggi, stampa, MAX_DIM */

using namespace std;

double calcola_norma_ricorsivo(int arr[], int n, int sum);
void dividi_ricorsivo(int source[], double dest[], double norma, int n);
double* normalizza(int source[], int dim);

int main() {
  int array[MAX_DIM];
  int dim;
  leggi(array, dim);
  double* normalizzato = normalizza(array, dim);
  cout << "Array normalizzato: " << endl;
  stampa(normalizzato, dim);
  return 0;
}

double calcola_norma_ricorsivo(int arr[], int n, int sum) {
  double res;
  if (n < 0) {
    res = sum;
  } else {
    sum += arr[n];
    res = calcola_norma_ricorsivo(arr, n - 1, sum);
  }
  return res;
}

void dividi_ricorsivo(int source[], double dest[], double norma, int n) {
```

```
  if (n >= 0) {
    dest[n] = source[n] / norma;
    dividi_ricorsivo(source, dest, norma, n - 1);
  }
}

double* normalizza(int source[], int dim) {
  double* normalizzato = new double[dim];
  double norma = calcola_norma_ricorsivo(source, dim - 1, 0);
  dividi_ricorsivo(source, normalizzato, norma, dim - 1);
  return normalizzato;
}
```

2 Scrivere nel file `esercizio2.cc`:

(a) una funzione **ricorsiva** *calcola_norma_ricorsivo* che, dato un array di *long* e la sua dimensione, restituisca un numero *double* corrispondente alla **norma ad uno** dell'array stesso;

(b) una funzione **ricorsiva** *normalizza* che, dato un array di *long* e la sua dimensione, restituisca un **nuovo array** contenente, per ogni elemento, l'elemento corrispondente del primo array diviso per la **norma ad uno** dell'array in input, calcolata con la funzione di cui al punto precedente.

Le operazioni di calcolo della norma e di divisione degli elementi dell'array per il valore della norma vanno eseguite tramite funzioni ricorsive, **pena l'annullamento dell'esercizio**.

Non è ammesso l'uso di cicli, variabili globali o static. È ammesso l'uso di funzioni ausiliarie, purché ricorsive. È consentito l'uso della funzione `sqrt` della libreria `cmath`.

È vietata ogni modifica alla funzione main pena l'annullamento dell'esercizio.

NOTA 1: La norma ad uno di un vettore è la somma dei degli elementi del vettore.

$$||x|| := \sum_{i=1}^{n} x_i$$

Ad esempio:

- Array in input: $[1, 2, 4]$
- Norma $= 1 + 2 + 4 = 7$
- Array normalizzato $= [1/7, 2/7, 4/7] = [0.142857, 0.285714, 0.571428]$

NOTA 2: I valori dell'array iniziale vanno inseriti utilizzando la funzione `leggi(...)` definita in `array.h` e `array.o`.
La stampa dell'array normalizzato va eseguita usando la funzione `stampa(...)` definita in `array.h` e `array.o`.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 **codice_A24.cc**

```
#include <iostream>
#include "array.h"     /* leggi, stampa, MAX_DIM */

using namespace std;

// Inserire qui sotto le DEFINIZIONI delle funzioni

int main() {
  long array[MAX_DIM];
  int dim;
  leggi(array, dim);
  double* normalizzato = normalizza(array, dim);
  cout << "Array normalizzato: " << endl;
  stampa(normalizzato, dim);
  return 0;
}

// Inserire qui sotto le DICHIARAZIONI delle funzioni
```

2 **soluzione_A24.cc**

```
#include <iostream>
#include "array.h"     /* leggi, stampa, MAX_DIM */

using namespace std;

double calcola_norma_ricorsivo(long arr[], int n, long sum);
void dividi_ricorsivo(long source[], double dest[], double norma, int n);
double* normalizza(long source[], int dim);

int main() {
  long array[MAX_DIM];
  int dim;
  leggi(array, dim);
  double* normalizzato = normalizza(array, dim);
  cout << "Array normalizzato: " << endl;
  stampa(normalizzato, dim);
  return 0;
}

double calcola_norma_ricorsivo(long arr[], int n, long sum) {
  double res;
  if (n < 0) {
    res = sum;
  } else {
    sum += arr[n];
    res = calcola_norma_ricorsivo(arr, n - 1, sum);
  }
  return res;
}

void dividi_ricorsivo(long source[], double dest[], double norma, int n) {
```

```
  if (n >= 0) {
    dest[n] = source[n] / norma;
    dividi_ricorsivo(source, dest, norma, n - 1);
  }
}

double* normalizza(long source[], int dim) {
  double* normalizzato = new double[dim];
  double norma = calcola_norma_ricorsivo(source, dim - 1, 0);
  dividi_ricorsivo(source, normalizzato, norma, dim - 1);
  return normalizzato;
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menù per gestire una pila di numeri `int`. Scrivere, in un nuovo file `stack.cc`, le definizioni delle funzioni dichiarate nello header file `stack.h` in modo tale che:

- `init` inizializzi la pila;
- `deinit` liberi la memoria utilizzata dalla pila;
- `empty` restituisca `FALSE` se la pila contiene almeno un valore, e `TRUE` altrimenti;
- `push` inserisca il valore passato come parametro nella pila;
- `pop` elimini il valore in testa alla pila, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `top` legga il valore in testa alla pila e lo memorizzi nella variabile passata come parametro, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `print` stampi a video il contenuto della pila, nell'ordine in cui i valori contenuti ne verrebbero estratti.

La pila deve essere implementata internamente come una lista concatenata i cui elementi sono allocati dinamicamente.

N.B.: Ad eccezione delle operazioni `print` e `deinit`, si ricorda che ogni altra operazione sulla pila deve avere costo costante, indipendentemente dal numero di valori contenuti nella struttura dati, **pena l'annullamento dell'esercizio**.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

3 **stack_main.cc**

```
using namespace std;
#include <iostream>

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    int val;

    init(s);

    do
    {
        cout << "Operazioni possibili:\n"
             << "u : push\n"
             << "o : pop\n"
             << "t : top\n"
             << "p : print\n"
             << "e : esci\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'u':
            cout << "Valore da inserire? ";
            cin >> val;
            push(s, val);
            break;
          case 'o':
            if (! pop(s))
                cout << "Stack vuoto!\n";
            else
                cout << "Pop ok!\n";
            break;
          case 't':
            if (! top(s, val))
                cout << "Stack vuoto!\n";
            else
                cout << "Top: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto dello stack: ";
            print(s);
            break;
          default:
            if (scelta != 'e') {
                cout << "Operazione non valida: " << scelta << endl;
            }
        }
```

```
    } while (scelta != 'e');

    deinit(s);

    return 0;
}
```

3 **stack.h**

```
#ifndef STACK_H
#define STACK_H

struct node {
  int val;
  node *next;
};

typedef node *stack;

enum retval {FALSE = 0, TRUE = 1};

void   init   (stack &s);
void   deinit (stack &s);
void   push   (stack &s, int val);
retval pop    (stack &s);
retval empty  (const stack &s);
retval top    (const stack &s, int &result);
void   print  (const stack &s);

#endif // STACK_H
```

3 **stack_.cc**

```
using namespace std;
#include <iostream>

#include "stack.h"


void init (stack &s)
{
    s = NULL;
}

void deinit (stack &s)
{
    node *n = s;
    while (n != NULL)
    {
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
```

```
    s = NULL;
}

retval empty (const stack &s)
{
    return (s == NULL ? TRUE : FALSE);
}

void push (stack &s, int  val)
{
    node *n = new node;

    n->val = val;
    n->next = s;
    s = n;
}

retval pop (stack &s)
{
    if (empty(s))
      return FALSE;

    node *first = s;
    s = s->next;
    delete first;

    return TRUE;
}

retval top (const stack &s, int &result)
{
    if (empty(s))
      return FALSE;

    result = s->val;
    return TRUE;
}

void print (const stack &s)
{
    node *n = s;
    while (n != NULL)
    {
        cout << n->val << " ";
        n = n->next;
    }
    cout << endl;
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menù per gestire una pila di numeri `long`. Scrivere, in un nuovo file `stack.cc`, le definizioni delle funzioni dichiarate nello header file `stack.h` in modo tale che:

- `init` inizializzi la pila;
- `deinit` liberi la memoria utilizzata dalla pila;
- `nempty` restituisca `TRUE` se la pila contiene almeno un valore, e `FALSE` altrimenti;
- `add` inserisca il valore passato come parametro nella pila;
- `shrink` elimini il valore in testa alla pila, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `first` legga il valore in testa alla pila e lo memorizzi nella variabile passata come parametro, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `print` stampi a video il contenuto della pila, nell'ordine in cui i valori contenuti ne verrebbero estratti.

La pila deve essere implementata internamente come una lista concatenata i cui elementi sono allocati dinamicamente.

N.B.: Ad eccezione delle operazioni `print` e `deinit`, si ricorda che ogni altra operazione sulla pila deve avere costo costante, indipendentemente dal numero di valori contenuti nella struttura dati, **pena l'annullamento dell'esercizio**.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

3 **stack_main.cc**

```
using namespace std;
#include <iostream>

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    long val;

    init(s);

    do
    {
        cout << "Operazioni possibili:\n"
             << "u : add\n"
             << "o : shrink\n"
             << "t : first\n"
             << "p : print\n"
             << "e : esci\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'u':
            cout << "Valore da inserire? ";
            cin >> val;
            add(s, val);
            break;
          case 'o':
            if (! shrink(s))
                cout << "Stack vuoto!\n";
            else
                cout << "Shrink ok!\n";
            break;
          case 't':
            if (! first(s, val))
                cout << "Stack vuoto!\n";
            else
                cout << "First: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto dello stack: ";
            print(s);
            break;
          default:
            if (scelta != 'e') {
                cout << "Operazione non valida: " << scelta << endl;
            }
        }
```

```
    } while (scelta != 'e');

    deinit(s);

    return 0;
}
```

3 stack.h

```
#ifndef STACK_H
#define STACK_H

struct node {
  long val;
  node *next;
};

typedef node *stack;

enum retval {FALSE = 0, TRUE = 1};

void   init   (stack &s);
void   deinit (stack &s);
retval shrink (stack &s);
void   add    (stack &s, long val);
retval first  (const stack &s, long &result);
void   print  (const stack &s);
retval nempty (const stack &s);

#endif // STACK_H
```

3 stack_.cc

```
using namespace std;
#include <iostream>

#include "stack.h"


void init (stack &s)
{
    s = NULL;
}

void deinit (stack &s)
{
    node *n = s;
    while (n != NULL)
    {
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
```

```
    s = NULL;
}

retval nempty (const stack &s)
{
    return (s == NULL ? FALSE : TRUE);
}

void add (stack &s, long  val)
{
    node *n = new node;

    n->val = val;
    n->next = s;
    s = n;
}

retval shrink (stack &s)
{
    if (!nempty(s))
      return FALSE;

    node *first = s;
    s = s->next;
    delete first;

    return TRUE;
}

retval first (const stack &s, long &result)
{
    if (!nempty(s))
      return FALSE;

    result = s->val;
    return TRUE;
}

void print (const stack &s)
{
    node *n = s;
    while (n != NULL)
    {
        cout << n->val << " ";
        n = n->next;
    }
    cout << endl;
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menù per gestire una pila di caratteri `char`. Scrivere, in un nuovo file `stack.cc`, le definizioni delle funzioni dichiarate nello header file `stack.h` in modo tale che:

- `init` inizializzi la pila;
- `deinit` liberi la memoria utilizzata dalla pila;
- `nempty` restituisca `TRUE` se la pila contiene almeno un carattere, e `FALSE` altrimenti;
- `push` inserisca il carattere passato come parametro nella pila;
- `pop` elimini il carattere in testa alla pila, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `top` legga il carattere in testa alla pila e lo memorizzi nella variabile passata come parametro, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `print` stampi a video il contenuto della pila, nell'ordine in cui i caratteri contenuti ne verrebbero estratti.

La pila deve essere implementata internamente come una lista concatenata i cui elementi sono allocati dinamicamente.

N.B.: Ad eccezione delle operazioni `print` e `deinit`, si ricorda che ogni altra operazione sulla pila deve avere costo costante, indipendentemente dal numero di valori contenuti nella struttura dati, **pena l'annullamento dell'esercizio**.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

3 **stack_main.cc**

```
using namespace std;
#include <iostream>

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    char val;

    init(s);

    do
    {
        cout << "Operazioni possibili:\n"
             << "u : push\n"
             << "o : pop\n"
             << "t : top\n"
             << "p : print\n"
             << "e : esci\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'u':
            cout << "Valore da inserire? ";
            cin >> val;
            push(s, val);
            break;
          case 'o':
            if (! pop(s))
                cout << "Stack vuoto!\n";
            else
                cout << "Pop ok!\n";
            break;
          case 't':
            if (! top(s, val))
                cout << "Stack vuoto!\n";
            else
                cout << "Top: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto dello stack: ";
            print(s);
            break;
          default:
            if (scelta != 'e') {
                cout << "Operazione non valida: " << scelta << endl;
            }
        }
```

```
    } while (scelta != 'e');

    deinit(s);

    return 0;
}
```

3 stack.h

```
#ifndef STACK_H
#define STACK_H

struct node {
  char val;
  node *next;
};

typedef node *stack;

enum retval {FALSE = 0, TRUE = 1};

void   init   (stack &s);
void   deinit (stack &s);
void   push   (stack &s, char val);
retval pop    (stack &s);
retval nempty (const stack &s);
retval top    (const stack &s, char &result);
void   print  (const stack &s);

#endif // STACK_H
```

3 stack_.cc

```
using namespace std;
#include <iostream>

#include "stack.h"


void init (stack &s)
{
    s = NULL;
}

void deinit (stack &s)
{
    node *n = s;
    while (n != NULL)
    {
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
```

```
    s = NULL;
}

retval nempty (const stack &s)
{
    return (s == NULL ? FALSE : TRUE);
}

void push (stack &s, char val)
{
    node *n = new node;

    n->val = val;
    n->next = s;
    s = n;
}

retval pop (stack &s)
{
    if (!nempty(s))
      return FALSE;

    node *first = s;
    s = s->next;
    delete first;

    return TRUE;
}

retval top (const stack &s, char &result)
{
    if (!nempty(s))
      return FALSE;

    result = s->val;
    return TRUE;
}

void print (const stack &s)
{
    node *n = s;
    while (n != NULL)
    {
        cout << n->val << " ";
        n = n->next;
    }
    cout << endl;
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menù per gestire una pila di numeri `double`. Scrivere, in un nuovo file `stack.cc`, le definizioni delle funzioni dichiarate nello header file `stack.h` in modo tale che:

- `init` inizializzi la pila;
- `deinit` liberi la memoria utilizzata dalla pila;
- `empty` restituisca `FALSE` se la pila contiene almeno un valore, e `TRUE` altrimenti;
- `add` inserisca il valore passato come parametro nella pila;
- `shrink` elimini il valore in testa alla pila, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `first` legga il valore in testa alla pila e lo memorizzi nella variabile passata come parametro, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `print` stampi a video il contenuto della pila, nell'ordine in cui i valori contenuti ne verrebbero estratti.

La pila deve essere implementata internamente come una lista concatenata i cui elementi sono allocati dinamicamente.

N.B.: Ad eccezione delle operazioni `print` e `deinit`, si ricorda che ogni altra operazione sulla pila deve avere costo costante, indipendentemente dal numero di valori contenuti nella struttura dati, **pena l'annullamento dell'esercizio**.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

3 **stack_main.cc**

```
using namespace std;
#include <iostream>

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    double val;

    init(s);

    do
    {
        cout << "Operazioni possibili:\n"
             << "u : add\n"
             << "o : shrink\n"
             << "t : first\n"
             << "p : print\n"
             << "e : esci\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'u':
            cout << "Valore da inserire? ";
            cin >> val;
            add(s, val);
            break;
          case 'o':
            if (! shrink(s))
                cout << "Stack vuoto!\n";
            else
                cout << "Shrink ok!\n";
            break;
          case 't':
            if (! first(s, val))
                cout << "Stack vuoto!\n";
            else
                cout << "First: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto dello stack: ";
            print(s);
            break;
          default:
            if (scelta != 'e') {
                cout << "Operazione non valida: " << scelta << endl;
            }
        }
```

```
    } while (scelta != 'e');

    deinit(s);

    return 0;
}
```

3 **stack.h**

```
#ifndef STACK_H
#define STACK_H

struct node {
  double val;
  node *next;
};

typedef node *stack;

enum retval {FALSE = 0, TRUE = 1};

void   init   (stack &s);
void   deinit (stack &s);
retval shrink (stack &s);
void   add    (stack &s, double val);
retval first  (const stack &s, double &result);
void   print  (const stack &s);
retval empty  (const stack &s);

#endif // STACK_H
```

3 **stack_.cc**

```
using namespace std;
#include <iostream>

#include "stack.h"


void init (stack &s)
{
    s = NULL;
}

void deinit (stack &s)
{
    node *n = s;
    while (n != NULL)
    {
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
```

```
    s = NULL;
}

retval empty (const stack &s)
{
    return (s == NULL ? TRUE : FALSE);
}

void add (stack &s, double  val)
{
    node *n = new node;

    n->val = val;
    n->next = s;
    s = n;
}

retval shrink (stack &s)
{
    if (empty(s))
      return FALSE;

    node *first = s;
    s = s->next;
    delete first;

    return TRUE;
}

retval first (const stack &s, double &result)
{
    if (empty(s))
      return FALSE;

    result = s->val;
    return TRUE;
}

void print (const stack &s)
{
    node *n = s;
    while (n != NULL)
    {
        cout << n->val << " ";
        n = n->next;
    }
    cout << endl;
}
```

4 L'algoritmo "BubbleSort" per ordinare un'array **A** di dimensione **dim** in ordine crescente funziona come segue: ogni coppia di elementi adiacenti dell'array viene comparata e, qualora siano nell'ordine sbagliato, vengono scambiati di posizione. L'algoritmo quindi scorre tutto l'array finché non vengono più eseguiti scambi, situazione che indica che l'array è completamente ordinato.

Si realizzi questo algoritmo nel file **esercizio4.cc** in forma COMPLETAMENTE RICORSIVA.

NOTA 1: Non è consentito utilizzare alcuna forma di ciclo, `break`, `continue`, `goto`, né l'uso di variabili globali o statiche o di funzioni di libreria.

NOTA 2: È consentito definire e utilizzare eventuali funzioni ausiliarie, purché a loro volta ricorsive e senza cicli.

NOTA 3: È vietato modificare la funzione main pena l'annullamento dell'esercizio.

VALUTAZIONE: questo esercizio permette di conseguire la lode se tutti gli esercizi precedenti sono corretti.

4 **codice_A41.cc**

```
using namespace std;
#include "array.h"

const int MAXDIM = 100;

// Inserire qui le DICHIARAZIONI delle funzioni

int main ()
{
  int myarray[MAXDIM];
  int dim;
  /*
  int myarray[MAXDIM] = {41,3,9,1,5,17,6,20,37,2,8,23,10,0,11,19};
  int dim = 16;
  bubblesort(myarray,dim);
  printarray(myarray,dim);
  */

  readarray(myarray,dim);
  bubblesort(myarray,dim);
  printarray(myarray,dim);
}

// Inserire qui le DEFINIZIONI delle funzioni
```

4 **soluzione_A41.cc**

```
using namespace std;
#include "array.h"

const int MAXDIM = 100;

void swap (int & a, int & b)
{
  int c = a;
  a = b;
  b = c;
}

// sposta il max da v[i] a v[k] in posizione v[k]
void push_max_upfront_rec (int v[],int i,int k)
{
  if (i<k) {
    if (v[i]>v[i+1])
      swap(v[i],v[i+1]);
    push_max_upfront_rec (v,i+1,k);
  }
}

// sposta il max da v[0] a v[k] in posizione v[k]
void push_max_upfront (int v[],int k)
{
```

```
  push_max_upfront_rec(v, 0, k);
}

void bubblesort (int v[],int n)
{
  if (n>0) {
    push_max_upfront(v, n-1);
    bubblesort(v,n-1);
  }
}

int main ()
{
  int myarray[MAXDIM];
  int dim;
  /*
  int myarray[MAXDIM] = {41,3,9,1,5,17,6,20,37,2,8,23,10,0,11,19};
  int dim = 16;
  bubblesort(myarray,dim);
  printarray(myarray,dim);
  */

  readarray(myarray,dim);
  bubblesort(myarray,dim);
  printarray(myarray,dim);
}
```

4 **array.cc**

```
using namespace std;
#include <iostream>
#include <iomanip>
#include "array.h"

void readarray(int v[],int & n)
{
  cout << "\ndimensione array?: ";
  cin >> n;
  for (int i=0;i<n;i++) {
    cout << "v[" << i << "]?: ";
    cin >> v[i];
  }
}

void printarray(int v[],int n)
{
  for (int i=0;i<n;i++) {
    cout << v[i] << " ";
  }
  cout << endl;
}
```