Terzo Appello di Programmazione I

15 Giugno 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 Scrivere un programma, nel file `esercizio1.cc`, che, presi come argomenti del `main` i nomi di due file, copi il primo file nel secondo correggendone la sintassi e generando in tal modo un testo "corretto" secondo le seguenti regole:

(a) la prima parola del testo deve iniziare con una lettera maiuscola;

(b) tutte le parole che seguono i seguenti caratteri: ".", "?" e "!", devono iniziare con una lettera maiuscola.

Se ad esempio l'eseguibile è `a.out`, il comando

```
./a.out testo testocorretto
```

creerà un nuovo file di nome `testocorretto` e vi copierà il contenuto del file dato `testo`, modificando le parole quando queste non verificano le regole descritte sopra. Nelle figure 1 e 2 un esempio di file `testo` e `testocorretto`.

filastrocca delle parole:
Fatevi avanti! chi ne vuole?
di parole ho la testa piena,
con dentro la "luna" e la "balena".
ci sono parole per gli amici:
Buon giorno, Buon anno, Siate felici!
parole belle e parole buone;
parole per ogni sorta di persone.
di G. Rodari.

Figura 1: `testo`

Filastrocca delle parole:
Fatevi avanti! Chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
Buon giorno, Buon anno, Siate felici!
Parole belle e parole buone;
parole per ogni sorta di persone.
Di G. Rodari.

Figura 2: `testocorretto`

NOTA 1: Per semplicità si assuma che il testo contenuto nel primo file inizi con un carattere alfabetico, non contenga "..." e che ".", "?" e "!" siano sempre preceduti da una parola e seguiti da uno spazio.

NOTA 2: Per semplicità si assuma che ogni parola contenuta nel testo del primo file abbia al massimo lunghezza 30 caratteri.

NOTA 3: È ammesso l'uso della funzione `strlen` della libreria `<cstring>`.

NOTA 4: il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). Per realizzare la conversione da caratteri minuscoli in maiuscoli, è vietato l'uso di tabelle o di 26 `if` o `switch-case`, uno per ogni carattere.

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

1 **soluzione_A11.cc**

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc,char* argv[]){

  fstream my_in, my_out;
  char tmp[31];

  if (argc!=3) {
    cout << "Usage: ./a.out <sourcefile>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  my_in >> tmp;
  if(!(tmp[0] >= 'A' && tmp[0] <= 'Z'))
    tmp[0] = tmp[0] + ('A'-'a');

  while(!my_in.eof()){
    my_out << tmp << " ";
    if(tmp[strlen(tmp)-1] == '.' || tmp[strlen(tmp)-1] == '?' || tmp[strlen(tmp)-1] == '!') {
      my_in >> tmp;
      if(!(tmp[0] >= 'A' && tmp[0] <= 'Z'))
        tmp[0] = tmp[0] + ('A'-'a');
    } else my_in >> tmp;
  }

  my_in.close();
  my_out.close();
  return(0);
}
```

1 Scrivere un programma, nel file `esercizio1.cc`, che, presi come argomenti del `main` i nomi di due file, copi il primo file nel secondo correggendone la sintassi e generando in tal modo un testo "corretto" secondo le seguenti regole:

(a) la prima parola del testo deve iniziare con una lettera maiuscola;

(b) tutte le parole che seguono i seguenti caratteri: ",", ";" e ":", devono iniziare con una lettera minuscola.

Se ad esempio l'eseguibile è `a.out`, il comando

```
./a.out testo testocorretto
```

creerà un nuovo file di nome `testocorretto` e vi copierà il contenuto del file dato `testo`, modificando le parole quando queste non verificano le regole descritte sopra. Nelle figure 1 e 2 un esempio di file `testo` e `testocorretto`.

filastrocca delle parole:
Fatevi avanti! chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
Buon giorno, buon anno, Siate felici!
Parole belle e parole buone;
Parole per ogni sorta di persone.
Di G. Rodari.

Figura 3: `testo`

Filastrocca delle parole:
fatevi avanti! chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
buon giorno, buon anno, siate felici!
Parole belle e parole buone;
parole per ogni sorta di persone.
Di G. Rodari.

Figura 4: `testocorretto`

NOTA 1: Per semplicità si assuma che il testo contenuto nel primo file inizi con un carattere alfabetico, non contenga "..." e che ".", "?" e "!" siano sempre preceduti da una parola e seguiti da uno spazio.

NOTA 2: Per semplicità si assuma che ogni parola contenuta nel testo del primo file abbia al massimo lunghezza 30 caratteri.

NOTA 3: È ammesso l'uso della funzione `strlen` della libreria `<cstring>`.

NOTA 4: il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). Per realizzare la conversione da caratteri minuscoli in maiuscoli, è vietato l'uso di tabelle o di 26 `if` o `switch-case`, uno per ogni carattere.

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

1 **soluzione_A12.cc**

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc,char* argv[]){

  fstream my_in, my_out;
  char tmp[31];

  if (argc!=3) {
    cout << "Usage: ./a.out <sourcefile>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  my_in >> tmp;
  if(tmp[0] >= 'a' && tmp[0] <= 'z')
    tmp[0] = tmp[0] + ('A'-'a');

  while(!my_in.eof()){
    my_out << tmp << " ";
    if(tmp[strlen(tmp)-1] == ',' || tmp[strlen(tmp)-1] == ';' || tmp[strlen(tmp)-1] == ':') {
      my_in >> tmp;
      if(!(tmp[0] >= 'a' && tmp[0] <= 'z'))
        tmp[0] = tmp[0] - ('A'-'a');
    } else my_in >> tmp;
  }

  my_in.close();
  my_out.close();
  return(0);
}
```

1 Scrivere un programma, nel file `esercizio1.cc`, che, presi come argomenti del `main` i nomi di due file, copi il primo file nel secondo correggendone la sintassi e generando in tal modo un testo "corretto" secondo le seguenti regole:

(a) la prima parola del testo deve iniziare con una lettera maiuscola;

(b) tutte le parole che seguono i seguenti caratteri: ".", "?" e "!", devono iniziare con una lettera maiuscola.

Se ad esempio l'eseguibile è `a.out`, il comando

```
./a.out testo testocorretto
```

creerà un nuovo file di nome `testocorretto` e vi copierà il contenuto del file dato `testo`, modificando le parole quando queste non verificano le regole descritte sopra. Nelle figure 1 e 2 un esempio di file `testo` e `testocorretto`.

filastrocca delle parole:
Fatevi avanti! chi ne vuole?
di parole ho la testa piena,
con dentro la "luna" e la "balena".
ci sono parole per gli amici:
Buon giorno, Buon anno, Siate felici!
parole belle e parole buone;
parole per ogni sorta di persone.
di G. Rodari.

Figura 5: `testo`

Filastrocca delle parole:
Fatevi avanti! Chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
Buon giorno, Buon anno, Siate felici!
Parole belle e parole buone;
parole per ogni sorta di persone.
Di G. Rodari.

Figura 6: `testocorretto`

NOTA 1: Per semplicità si assuma che il testo contenuto nel primo file inizi con un carattere alfabetico, non contenga "..." e che ".", "?" e "!" siano sempre preceduti da una parola e seguiti da uno spazio.

NOTA 2: Per semplicità si assuma che ogni parola contenuta nel testo del primo file abbia al massimo lunghezza 30 caratteri.

NOTA 3: È ammesso l'uso della funzione `strlen` della libreria `<cstring>`.

NOTA 4: il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). Per realizzare la conversione da caratteri minuscoli in maiuscoli, è vietato l'uso di tabelle o di 26 `if` o `switch-case`, uno per ogni carattere.

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

1 **soluzione_A13.cc**

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc,char* argv[]){

  fstream my_in, my_out;
  char tmp[31];

  if (argc!=3) {
    cout << "Usage: ./a.out <sourcefile>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  my_in >> tmp;
  if(!(tmp[0] >= 'A' && tmp[0] <= 'Z'))
    tmp[0] = tmp[0] + ('A'-'a');

  while(!my_in.eof()){
    my_out << tmp << " ";
    if(tmp[strlen(tmp)-1] == '.' || tmp[strlen(tmp)-1] == '?' || tmp[strlen(tmp)-1] == '!') {
      my_in >> tmp;
      if(!(tmp[0] >= 'A' && tmp[0] <= 'Z'))
        tmp[0] = tmp[0] + ('A'-'a');
    } else my_in >> tmp;
  }

  my_in.close();
  my_out.close();
  return(0);
}
```

1 Scrivere un programma, nel file `esercizio1.cc`, che, presi come argomenti del `main` i nomi di due file, copi il primo file nel secondo correggendone la sintassi e generando in tal modo un testo "corretto" secondo le seguenti regole:

(a) la prima parola del testo deve iniziare con una lettera maiuscola;

(b) tutte le parole che seguono i seguenti caratteri: ",", ";" e ":", devono iniziare con una lettera minuscola.

Se ad esempio l'eseguibile è `a.out`, il comando

```
./a.out testo testocorretto
```

creerà un nuovo file di nome `testocorretto` e vi copierà il contenuto del file dato `testo`, modificando le parole quando queste non verificano le regole descritte sopra. Nelle figure 1 e 2 un esempio di file `testo` e `testocorretto`.

filastrocca delle parole:
Fatevi avanti! chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
Buon giorno, buon anno, Siate felici!
Parole belle e parole buone;
Parole per ogni sorta di persone.
Di G. Rodari.

Figura 7: `testo`

Filastrocca delle parole:
fatevi avanti! chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
buon giorno, buon anno, siate felici!
Parole belle e parole buone;
parole per ogni sorta di persone.
Di G. Rodari.

Figura 8: `testocorretto`

NOTA 1: Per semplicità si assuma che il testo contenuto nel primo file inizi con un carattere alfabetico, non contenga "..." e che ".", "?" e "!" siano sempre preceduti da una parola e seguiti da uno spazio.

NOTA 2: Per semplicità si assuma che ogni parola contenuta nel testo del primo file abbia al massimo lunghezza 30 caratteri.

NOTA 3: È ammesso l'uso della funzione `strlen` della libreria `<cstring>`.

NOTA 4: il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). Per realizzare la conversione da caratteri minuscoli in maiuscoli, è vietato l'uso di tabelle o di 26 `if` o `switch-case`, uno per ogni carattere.

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

1 **soluzione_A14.cc**

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <cstring>
using namespace std;

int main(int argc,char* argv[]){

  fstream my_in, my_out;
  char tmp[31];

  if (argc!=3) {
    cout << "Usage: ./a.out <sourcefile>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  my_in >> tmp;
  if(tmp[0] >= 'a' && tmp[0] <= 'z')
    tmp[0] = tmp[0] + ('A'-'a');

  while(!my_in.eof()){
    my_out << tmp << " ";
    if(tmp[strlen(tmp)-1] == ',' || tmp[strlen(tmp)-1] == ';' || tmp[strlen(tmp)-1] == ':') {
      my_in >> tmp;
      if(!(tmp[0] >= 'a' && tmp[0] <= 'z'))
        tmp[0] = tmp[0] - ('A'-'a');
    } else my_in >> tmp;
  }

  my_in.close();
  my_out.close();
  return(0);
}
```

2 Il programma nel file **esercizio2.cc** ha lo scopo di analizzare il contenuto di una matrice di numeri **interi**, precedentemente inizializzata con valori casuali, modificandone il contenuto secondo un certo criterio; in particolare, il programma:

(a) crea e popola una matrice dinamica di numeri interi, inizializzata con valori casuali utilizzando la funzione `generaMatrice`, definita all'interno dei file `util.h` e `util.o` inclusi;

(b) ne stampa il contenuto a video utilizzando la procedura `stampaMatrice`, anch'essa definita nei file `util.h` e `util.o` di cui sopra;

(c) crea una nuova matrice dinamica delle stesse dimensioni di quella generata al punto (**a**), e salva in ogni cella il valore assoluto del contenuto della corrispondente cella della prima matrice;

(d) stampa il contenuto a video della nuova matrice utilizzando la procedura `stampaMatrice`;

Si richiede di completare il file **esercizio2.cc** implementando la funzione `modificaMatrice`, che riceve in ingresso una matrice di interi, il numero di righe e di colonne della matrice da analizzare e ritorna una nuova matrice, dove i numeri sono uguali a quelli della matrice di input, trasformati secondo il criterio enunciato al punto (**c**) del precedente elenco.

Se la funzione `generaMatrice` restituisce una matrice così fatta:

```
 1 12 -10  9  3
 4  -5  7  8 18
81  0  0  -3  4
 4  5  -6  7  8
```

la matrice "trasformata" sarà:

```
 1 12  10  9  3
 4  5  7  8 18
81  0  0  3  4
 4  5  6  7  8
```

NOTA 1: La funzione `modificaMatrice` non deve modificare in alcun modo il contenuto della matrice in ingresso.

NOTA 2: Non è necessario gestire esplicitamente eventuali errori dovuti all'esaurimento della memoria dinamica disponibile.

NOTA 3: Qualora ritenuto necessario è possibile dichiarare e definire funzioni ausiliarie; non è tuttavia consentito impiegare funzioni di libreria ad eccezione delle funzioni `generaMatrice` e `stampaMatrice`.

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

2 **codice_A21.cc**

```
#include <iostream>
#include "util.h"

using namespace std;

// Inserire qui la DICHIARAZIONE della funzione modificaMatrice

const int NR = 10;
const int NC = 10;

int main(int argc, char * argv[]) {
  int** matriceOriginale = generaMatrice(NR, NC);

  cout << "Matrice originale:" << endl;
  stampaMatrice(matriceOriginale, NR, NC);
  cout << endl;

  int** matriceModificata = modificaMatrice(matriceOriginale, NR, NC);

  cout << "Matrice modificata:" << endl;
  stampaMatrice(matriceModificata, NR, NC);
  cout << endl;

  return 0;
}

// Inserire qui la DEFINIZIONE della funzione modificaMatrice
```

2 **soluzione_A21.cc**

```
#include <iostream>
#include "util.h"

using namespace std;

int** modificaMatrice(int** matrice, int nc, int nr);

const int NR = 10;
const int NC = 10;

int main(int argc, char * argv[]) {
  int** matriceOriginale = generaMatrice(NR, NC);

  cout << "Matrice originale:" << endl;
  stampaMatrice(matriceOriginale, NR, NC);
  cout << endl;

  int** matriceModificata = modificaMatrice(matriceOriginale, NR, NC);

  cout << "Matrice modificata:" << endl;
  stampaMatrice(matriceModificata, NR, NC);
  cout << endl;
```

```
  return 0;
}

int** modificaMatrice(int** matrice, int nc, int nr) {
  // Alloca dinamicamente la matrice da ritornare
  int** daRitornare = new int* [nr];
  for(int i = 0; i < nc; i++) {
    daRitornare[i] = new int [nc];
  }
  // Copia i dati da matrice originale a matrice
  // da ritornare, modificandoli
  for(int i = 0; i < nr; i++) {
    for(int j = 0; j < nc; j++) {
      // Valore assoluto
      daRitornare[i][j] = matrice[i][j] > 0 ? matrice[i][j] : -matrice[i][j];
    }
  }
  // Ritorna la matrice
  return daRitornare;
}
```

2 Il programma nel file **esercizio2.cc** ha lo scopo di analizzare il contenuto di una matrice di numeri **float**, precedentemente inizializzata con valori casuali, modificandone il contenuto secondo un certo criterio; in particolare, il programma:

(a) crea e popola una matrice dinamica di numeri float, inizializzata con valori casuali utilizzando la funzione `generaMatrice`, definita all'interno dei file `util.h` e `util.o` inclusi;

(b) ne stampa il contenuto a video utilizzando la procedura `stampaMatrice`, anch'essa definita nei file `util.h` e `util.o` di cui sopra;

(c) crea una nuova matrice dinamica delle stesse dimensioni di quella generata al punto (**a**), e salva in ogni cella il cubo (x^3) del contenuto della corrispondente cella della prima matrice;

(d) stampa il contenuto a video della nuova matrice utilizzando la procedura `stampaMatrice`;

Si richiede di completare il file **esercizio2.cc** implementando la funzione `modificaMatrice`, che riceve in ingresso una matrice di float, il numero di righe e di colonne della matrice da analizzare e ritorna una nuova matrice, dove i numeri sono uguali a quelli della matrice di input, trasformati secondo il criterio enunciato al punto (**c**) del precedente elenco.

Se la funzione `generaMatrice` restituisce una matrice così fatta:

```
 1.5   12.0  -10.9   9.67    3.45
 4.12  -5.9   70    85      18.9
81.2    0.1    0    -3.496   4.1
 4      5.7   -6.5   7.12    8.339
```

la matrice "trasformata" sarà:

```
1.157625  1728  -1295.029 904.231063  41.063625
69.934528 -205.379  343000  614125  6751.269
535387.328  0.001 0 -42.72816794  68.921
64  185.193 -274.625  360.944128  579.8850622
```

NOTA 1: La funzione `modificaMatrice` non deve modificare in alcun modo il contenuto della matrice in ingresso.

NOTA 2: Non è necessario gestire esplicitamente eventuali errori dovuti all'esaurimento della memoria dinamica disponibile.

NOTA 3: Qualora ritenuto necessario è possibile dichiarare e definire funzioni ausiliarie; non è tuttavia consentito impiegare funzioni di libreria ad eccezione delle funzioni `generaMatrice` e `stampaMatrice`.

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

2 **codice_A22.cc**

```
#include <iostream>
#include "util.h"

using namespace std;

// Inserire qui la DICHIARAZIONE della funzione modificaMatrice

const int NR = 10;
const int NC = 10;

int main(int argc, char * argv[]) {
  float** matriceOriginale = generaMatrice(NR, NC);

  cout << "Matrice originale:" << endl;
  stampaMatrice(matriceOriginale, NR, NC);
  cout << endl;

  float** matriceModificata = modificaMatrice(matriceOriginale, NR, NC);

  cout << "Matrice modificata:" << endl;
  stampaMatrice(matriceModificata, NR, NC);
  cout << endl;

  return 0;
}

// Inserire qui la DEFINIZIONE della funzione modificaMatrice
```

2 **soluzione_A22.cc**

```
#include <iostream>
#include "util.h"

using namespace std;

float** modificaMatrice(float** matrice, int nc, int nr);

const int NR = 10;
const int NC = 10;

int main(int argc, char * argv[]) {
  float** matriceOriginale = generaMatrice(NR, NC);

  cout << "Matrice originale:" << endl;
  stampaMatrice(matriceOriginale, NR, NC);
  cout << endl;

  float** matriceModificata = modificaMatrice(matriceOriginale, NR, NC);

  cout << "Matrice modificata:" << endl;
  stampaMatrice(matriceModificata, NR, NC);
  cout << endl;
```

```
  return 0;
}

float** modificaMatrice(float** matrice, int nc, int nr) {
  // Alloca dinamicamente la matrice da ritornare
  float** daRitornare = new float* [nr];
  for(int i = 0; i < nc; i++) {
    daRitornare[i] = new float [nc];
  }
  // Copia i dati da matrice originale a matrice
  // da ritornare, modificandoli
  for(int i = 0; i < nr; i++) {
    for(int j = 0; j < nc; j++) {
      // Cubo
      daRitornare[i][j] = matrice[i][j] * matrice[i][j] * matrice[i][j];
    }
  }
  // Ritorna la matrice
  return daRitornare;
}
```

2 Il programma nel file **esercizio2.cc** ha lo scopo di analizzare il contenuto di una matrice di numeri **interi**, precedentemente inizializzata con valori casuali, modificandone il contenuto secondo un certo criterio; in particolare, il programma:

(a) crea e popola una matrice dinamica di numeri interi, inizializzata con valori casuali utilizzando la funzione `generaMatrice`, definita all'interno dei file `util.h` e `util.o` inclusi;

(b) ne stampa il contenuto a video utilizzando la procedura `stampaMatrice`, anch'essa definita nei file `util.h` e `util.o` di cui sopra;

(c) crea una nuova matrice dinamica delle stesse dimensioni di quella generata al punto (**a**), e salva in ogni cella il contenuto della corrispondente cella della prima matrice, se è un numero pari, altrimenti vi aggiunge 1;

(d) stampa il contenuto a video della nuova matrice utilizzando la procedura `stampaMatrice`;

Si richiede di completare il file **esercizio2.cc** implementando la funzione `modificaMatrice`, che riceve in ingresso una matrice di interi, il numero di righe e di colonne della matrice da analizzare e ritorna una nuova matrice, dove i numeri sono uguali a quelli della matrice di input, trasformati secondo il criterio enunciato al punto (**c**) del precedente elenco.

Se la funzione `generaMatrice` restituisce una matrice così fatta:

```
 1 12 -10  9  3
 4  -5  7  8 18
81  0  0  -3  4
 4  5  -6  7  8
```

la matrice "trasformata" sarà:

```
 2 12 -10  10  4
 4  -4  8  8 18
82  0  0  -2  4
 4  6  -6  8  8
```

NOTA 1: La funzione `modificaMatrice` non deve modificare in alcun modo il contenuto della matrice in ingresso.

NOTA 2: Non è necessario gestire esplicitamente eventuali errori dovuti all'esaurimento della memoria dinamica disponibile.

NOTA 3: Qualora ritenuto necessario è possibile dichiarare e definire funzioni ausiliarie; non è tuttavia consentito impiegare funzioni di libreria ad eccezione delle funzioni `generaMatrice` e `stampaMatrice`.

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

2 **codice_A23.cc**

```
#include <iostream>
#include "util.h"

using namespace std;

// Inserire qui la DICHIARAZIONE della funzione modificaMatrice

const int NR = 10;
const int NC = 10;

int main(int argc, char * argv[]) {
  int** matriceOriginale = generaMatrice(NR, NC);

  cout << "Matrice originale:" << endl;
  stampaMatrice(matriceOriginale, NR, NC);
  cout << endl;

  int** matriceModificata = modificaMatrice(matriceOriginale, NR, NC);

  cout << "Matrice modificata:" << endl;
  stampaMatrice(matriceModificata, NR, NC);
  cout << endl;

  return 0;
}

// Inserire qui la DEFINIZIONE della funzione modificaMatrice
```

2 **soluzione_A23.cc**

```
#include <iostream>
#include "util.h"

using namespace std;

int** modificaMatrice(int** matrice, int nc, int nr);

const int NR = 10;
const int NC = 10;

int main(int argc, char * argv[]) {
  int** matriceOriginale = generaMatrice(NR, NC);

  cout << "Matrice originale:" << endl;
  stampaMatrice(matriceOriginale, NR, NC);
  cout << endl;

  int** matriceModificata = modificaMatrice(matriceOriginale, NR, NC);

  cout << "Matrice modificata:" << endl;
  stampaMatrice(matriceModificata, NR, NC);
  cout << endl;
```

```
  return 0;
}

int** modificaMatrice(int** matrice, int nc, int nr) {
  // Alloca dinamicamente la matrice da ritornare
  int** daRitornare = new int* [nr];
  for(int i = 0; i < nc; i++) {
    daRitornare[i] = new int [nc];
  }
  // Copia i dati da matrice originale a matrice
  // da ritornare, modificandoli
  for(int i = 0; i < nr; i++) {
    for(int j = 0; j < nc; j++) {
      // Rende pari i numeri dispari aggiungendo 1;
      // non fa nulla altrimenti
      daRitornare[i][j] = matrice[i][j] % 2 == 0 ? matrice[i][j] : matrice[i][j] + 1;
    }
  }
  // Ritorna la matrice
  return daRitornare;
}
```

2 Il programma nel file **esercizio2.cc** ha lo scopo di analizzare il contenuto di una matrice di numeri **float**, precedentemente inizializzata con valori casuali, modificandone il contenuto secondo un certo criterio; in particolare, il programma:

(a) crea e popola una matrice dinamica di numeri float, inizializzata con valori casuali utilizzando la funzione `generaMatrice`, definita all'interno dei file `util.h` e `util.o` inclusi;

(b) ne stampa il contenuto a video utilizzando la procedura `stampaMatrice`, anch'essa definita nei file `util.h` e `util.o` di cui sopra;

(c) crea una nuova matrice dinamica delle stesse dimensioni di quella generata al punto (**a**), e salva in ogni cella l'inverso $(1/x)$ del contenuto della corrispondente cella della prima matrice, se diverso da zero, e "0" altrimenti;

(d) stampa il contenuto a video della nuova matrice utilizzando la procedura `stampaMatrice`;

Si richiede di completare il file **esercizio2.cc** implementando la funzione `modificaMatrice`, che riceve in ingresso una matrice di float, il numero di righe e di colonne della matrice da analizzare e ritorna una nuova matrice, dove i numeri sono uguali a quelli della matrice di input, trasformati secondo il criterio enunciato al punto (**c**) del precedente elenco.

Se la funzione `generaMatrice` restituisce una matrice così fatta:

```
 1.5   12.0  -10.9   9.67    3.45
 4.12  -5.9   70    85      18.9
81.2    0.1    0    -3.496   4.1
 4      5.7   -6.5   7.12    8.339
```

la matrice "trasformata" sarà:

```
0.6666666667  0.08333333333 -0.09174311927  0.1034126163  0.2898550725
0.2427184466  -0.1694915254 0.01428571429 0.01176470588 0.05291005291
0.01231527094 10  0 -0.2860411899 0.243902439
0.25  0.1754385965  -0.1538461538 0.1404494382  0.1199184555
```

NOTA 1: La funzione `modificaMatrice` non deve modificare in alcun modo il contenuto della matrice in ingresso.

NOTA 2: Non è necessario gestire esplicitamente eventuali errori dovuti all'esaurimento della memoria dinamica disponibile.

NOTA 3: Qualora ritenuto necessario è possibile dichiarare e definire funzioni ausiliarie; non è tuttavia consentito impiegare funzioni di libreria ad eccezione delle funzioni `generaMatrice` e `stampaMatrice`.

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

2 **codice_A24.cc**

```
#include <iostream>
#include "util.h"

using namespace std;

// Inserire qui la DICHIARAZIONE della funzione modificaMatrice

const int NR = 10;
const int NC = 10;

int main(int argc, char * argv[]) {
  float** matriceOriginale = generaMatrice(NR, NC);

  cout << "Matrice originale:" << endl;
  stampaMatrice(matriceOriginale, NR, NC);
  cout << endl;

  float** matriceModificata = modificaMatrice(matriceOriginale, NR, NC);

  cout << "Matrice modificata:" << endl;
  stampaMatrice(matriceModificata, NR, NC);
  cout << endl;

  return 0;
}

// Inserire qui la DEFINIZIONE della funzione modificaMatrice
```

2 **soluzione_A24.cc**

```
#include <iostream>
#include "util.h"

using namespace std;

float** modificaMatrice(float** matrice, int nc, int nr);

const int NR = 10;
const int NC = 10;

int main(int argc, char * argv[]) {
  float** matriceOriginale = generaMatrice(NR, NC);

  cout << "Matrice originale:" << endl;
  stampaMatrice(matriceOriginale, NR, NC);
  cout << endl;

  float** matriceModificata = modificaMatrice(matriceOriginale, NR, NC);

  cout << "Matrice modificata:" << endl;
  stampaMatrice(matriceModificata, NR, NC);
  cout << endl;
```

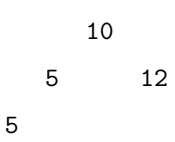
```
  return 0;
}

float** modificaMatrice(float** matrice, int nc, int nr) {
  // Alloca dinamicamente la matrice da ritornare
  float** daRitornare = new float* [nr];
  for(int i = 0; i < nc; i++) {
    daRitornare[i] = new float [nc];
  }
  // Copia i dati da matrice originale a matrice
  // da ritornare, modificandoli
  for(int i = 0; i < nr; i++) {
    for(int j = 0; j < nc; j++) {
      // Inverso se non nullo
      daRitornare[i][j] = matrice[i][j] == 0 ? 0 : 1 / matrice[i][j];
    }
  }
  // Ritorna la matrice
  return daRitornare;
}
```

3 Nel file `albero_main.cc` è definita la funzione `main` che contiene un menù per gestire un albero binario di ricerca di `int`. Scrivere nel file `albero.cc` le definizioni delle funzioni dichiarate nel file di header `albero.h` in modo tale che:

- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **crescente** e se l'elemento è già presente deve essere inserito a **sinistra**. Esempio: l'inserimento dei seguenti valori:

  ```
  10  5  12  5
  ```

  deve produrre il seguente albero:

- `cerca` cerchi nell'albero l'elemento passato in input, resituendo `VERO` se l'elemento è presente, e `FALSO` altrimenti;
- `stampa` stampi a video il contenuto dell'albero, in ordine **crescente**. Esempio: l'albero qui sopra deve essere stampato come:

  ```
  5  5  10  12
  ```

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

3 albero_main.cc

```
#include <iostream>
#include "albero.h"

using namespace std;

int main() {
  char res;
  Albero albero;
  int val;
  inizializza(albero);
  do {
    cout << "\nOperazioni possibili:\n"
         << " Inserimento (i)\n"
         << " Ricerca (r)\n"
         << " Stampa ordinata (s)\n"
         << " Fine (f)\n";
    cout << "Operazione selezionata: ";
    cin >> res;
    switch (res) {
    case 'i':
      cout << "Valore : ";
      cin >> val;
      if (inserisci(albero, val) == FALSO) {
        cout << "Valore gia' presente!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (cerca(albero, val) == VERO) {
        cout << "Valore presente: " << val << endl;
      } else {
        cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (vuoto(albero) == VERO) {
        cout << "Albero vuoto!" << endl;
      } else {
        stampa(albero);
      }
      break;
    case 'f':
      break;
    default:
      cout << "Opzione errata\n";
    }
  } while (res != 'f');

  return 0;
}
```

3 **albero.h**

```
// -*- C++ -*-
#ifndef ALBERO_H
#define ALBERO_H

struct Nodo {
    int val;
    Nodo *sx;
    Nodo *dx;
};

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, int val);
boolean cerca(const Albero &t, int val);
void stampa(const Albero &t);

#endif
```

3 **albero_.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

void inizializza(Albero &a) {
  a = NULL;
}

boolean vuoto(const Albero &a) {
  return (a == NULL) ? VERO : FALSO;
}

boolean inserisci(Albero &a, int val) {
  // Caso base
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a == NULL) {
       return FALSO;
    }
    a->val = val;
    a->sx = a->dx = NULL;
    return VERO;
  }
  // Caso ricorsivo. Controllo se scendere a sinistra o a destra
  if (val <= a->val) {
    // Scendo a sinistra
    return inserisci(a->sx, val);
```

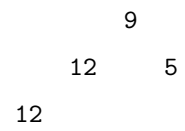
```
  } else if (val > a->val) {
    // Scendo a destra
    return inserisci(a->dx, val);
  }
}

boolean cerca(const Albero &a, int val) {
  if (vuoto(a) == VERO) {
    return FALSO;
  } else if (val == a->val) {
    // Trovato
    return VERO;
  } else if (val < a->val) {
    // Scendo a sinistra
    return cerca(a->sx, val);
  } else {
    // Scendo a destra
    return cerca(a->dx, val);
  }
}

void stampa(const Albero &a) {
  if (vuoto(a) == FALSO) {
    // Prima stampo gli elementi MINORI di a->val (cioe' quelli a sx)
    stampa(a->sx);
    // Poi stampo a->val
    cout << a-> val << ' ';
    // Poi stampo gli elementi MAGGIORI (cioe' quelli a dx)
    stampa(a->dx);
  }
}
```

3 Nel file `albero_main.cc` è definita la funzione `main` che contiene un menù per gestire un albero binario di ricerca di `int`. Scrivere nel file `albero.cc` le definizioni delle funzioni dichiarate nel file di header `albero.h` in modo tale che:

- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **decrescente** e se l'elemento è già presente deve essere inserito a **sinistra**. Esempio: l'inserimento dei seguenti valori:

  ```
  9  12  5  12
  ```

  deve produrre il seguente albero:

- `cerca` cerchi nell'albero l'elemento passato in input, resituendo `VERO` se l'elemento è presente, e `FALSO` altrimenti;
- `stampa` stampi a video il contenuto dell'albero, in ordine **crescente**. Esempio: l'albero qui sopra deve essere stampato come:

  ```
  5  5  9  12
  ```

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

3 **albero_main.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

int main() {
  char res;
  Albero albero;
  int val;
  inizializza(albero);
  do {
    cout << "\nOperazioni possibili:\n"
         << " Inserimento (i)\n"
         << " Ricerca (r)\n"
         << " Stampa ordinata (s)\n"
         << " Fine (f)\n";
    cout << "Operazione selezionata: ";
    cin >> res;
    switch (res) {
    case 'i':
      cout << "Valore : ";
      cin >> val;
      if (inserisci(albero, val) == FALSO) {
        cout << "Valore gia' presente!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (cerca(albero, val) == VERO) {
        cout << "Valore presente: " << val << endl;
      } else {
        cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (vuoto(albero) == VERO) {
        cout << "Albero vuoto!" << endl;
      } else {
        stampa(albero);
      }
      break;
    case 'f':
      break;
    default:
      cout << "Opzione errata\n";
    }
  } while (res != 'f');

  return 0;
}
```

3 **albero.h**

```
// -*- C++ -*-
#ifndef ALBERO_H
#define ALBERO_H

struct Nodo {
    int val;
    Nodo *sx;
    Nodo *dx;
};

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, int val);
boolean cerca(const Albero &t, int val);
void stampa(const Albero &t);

#endif
```

3 **albero_.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

void inizializza(Albero &a) {
  a = NULL;
}

boolean vuoto(const Albero &a) {
  return (a == NULL) ? VERO : FALSO;
}

boolean inserisci(Albero &a, int val) {
  // Caso base
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a == NULL) {
       return FALSO;
    }
    a->val = val;
    a->sx = a->dx = NULL;
    return VERO;
  }
  // Caso ricorsivo. Controllo se scendere a sinistra o a destra
  if (val < a->val) {
    // Scendo a destra
    return inserisci(a->dx, val);
```

```
  } else if (val >= a->val) {
    // Scendo a sinistra
    return inserisci(a->sx, val);
  }
}

boolean cerca(const Albero &a, int val) {
  if (vuoto(a) == VERO) {
    return FALSO;
  } else if (val == a->val) {
    // Trovato
    return VERO;
  } else if (val < a->val) {
    // Scendo a destra
    return cerca(a->dx, val);
  } else {
    // Scendo a sinistra
    return cerca(a->sx, val);
  }
}

void stampa(const Albero &a) {
  if (vuoto(a) == FALSO) {
    // Prima stampo gli elementi MINORI di a->val (cioe' quelli a dx)
    stampa(a->dx);
    // Poi stampo a->val
    cout << a-> val << ' ';
    // Poi stampo gli elementi MAGGIORI (cioe' quelli a sx)
    stampa(a->sx);
  }
}
```

3 Nel file `albero_main.cc` è definita la funzione `main` che contiene un menù per gestire un albero binario di ricerca di `int`. Scrivere nel file `albero.cc` le definizioni delle funzioni dichiarate nel file di header `albero.h` in modo tale che:

- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **crescente** e se l'elemento è già presente deve essere inserito a **destra**. Esempio: l'inserimento dei seguenti valori:

  ```
  7  8  6  8
  ```

  deve produrre il seguente albero:

  ```
      7
  6     8
          8
  ```

- `cerca` cerchi nell'albero l'elemento passato in input, resituendo `VERO` se l'elemento è presente, e `FALSO` altrimenti;
- `stampa` stampi a video il contenuto dell'albero, in ordine **crescente**. Esempio: l'albero qui sopra deve essere stampato come:

  ```
  6  7  8  8
  ```

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

3 **albero_main.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

int main() {
  char res;
  Albero albero;
  int val;
  inizializza(albero);
  do {
    cout << "\nOperazioni possibili:\n"
         << " Inserimento (i)\n"
         << " Ricerca (r)\n"
         << " Stampa ordinata (s)\n"
         << " Fine (f)\n";
    cout << "Operazione selezionata: ";
    cin >> res;
    switch (res) {
    case 'i':
      cout << "Valore : ";
      cin >> val;
      if (inserisci(albero, val) == FALSO) {
        cout << "Valore gia' presente!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (cerca(albero, val) == VERO) {
        cout << "Valore presente: " << val << endl;
      } else {
        cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (vuoto(albero) == VERO) {
        cout << "Albero vuoto!" << endl;
      } else {
        stampa(albero);
      }
      break;
    case 'f':
      break;
    default:
      cout << "Opzione errata\n";
    }
  } while (res != 'f');

  return 0;
}
```

3 **albero.h**

```
// -*- C++ -*-
#ifndef ALBERO_H
#define ALBERO_H

struct Nodo {
    int val;
    Nodo *sx;
    Nodo *dx;
};

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, int val);
boolean cerca(const Albero &t, int val);
void stampa(const Albero &t);

#endif
```

3 **albero_.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

void inizializza(Albero &a) {
  a = NULL;
}

boolean vuoto(const Albero &a) {
  return (a == NULL) ? VERO : FALSO;
}

boolean inserisci(Albero &a, int val) {
  // Caso base
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a == NULL) {
       return FALSO;
    }
    a->val = val;
    a->sx = a->dx = NULL;
    return VERO;
  }
  // Caso ricorsivo. Controllo se scendere a sinistra o a destra
  if (val < a->val) {
    // Scendo a sinistra
    return inserisci(a->sx, val);
```

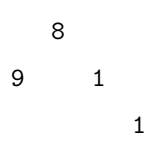
```
  } else if (val >= a->val) {
    // Scendo a destra
    return inserisci(a->dx, val);
  }
}

boolean cerca(const Albero &a, int val) {
  if (vuoto(a) == VERO) {
    return FALSO;
  } else if (val == a->val) {
    // Trovato
    return VERO;
  } else if (val < a->val) {
    // Scendo a sinistra
    return cerca(a->sx, val);
  } else {
    // Scendo a destra
    return cerca(a->dx, val);
  }
}

void stampa(const Albero &a) {
  if (vuoto(a) == FALSO) {
    // Prima stampo gli elementi MINORI di a->val (cioe' quelli a sx)
    stampa(a->sx);
    // Poi stampo a->val
    cout << a-> val << ' ';
    // Poi stampo gli elementi MAGGIORI (cioe' quelli a dx)
    stampa(a->dx);
  }
}
```

3 Nel file `albero_main.cc` è definita la funzione `main` che contiene un menù per gestire un albero binario di ricerca di `int`. Scrivere nel file `albero.cc` le definizioni delle funzioni dichiarate nel file di header `albero.h` in modo tale che:

- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **decrescente** e se l'elemento è già presente deve essere inserito a **destra**. Esempio: l'inserimento dei seguenti valori:

  ```
  8  9  1  1
  ```

  deve produrre il seguente albero:

- `cerca` cerchi nell'albero l'elemento passato in input, resituendo `VERO` se l'elemento è presente, e `FALSO` altrimenti;
- `stampa` stampi a video il contenuto dell'albero, in ordine **crescente**. Esempio: l'albero qui sopra deve essere stampato come:

  ```
  1  1  8  9
  ```

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

3 **albero_main.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

int main() {
  char res;
  Albero albero;
  int val;
  inizializza(albero);
  do {
    cout << "\nOperazioni possibili:\n"
         << " Inserimento (i)\n"
         << " Ricerca (r)\n"
         << " Stampa ordinata (s)\n"
         << " Fine (f)\n";
    cout << "Operazione selezionata: ";
    cin >> res;
    switch (res) {
    case 'i':
      cout << "Valore : ";
      cin >> val;
      if (inserisci(albero, val) == FALSO) {
        cout << "Valore gia' presente!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (cerca(albero, val) == VERO) {
        cout << "Valore presente: " << val << endl;
      } else {
        cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (vuoto(albero) == VERO) {
        cout << "Albero vuoto!" << endl;
      } else {
        stampa(albero);
      }
      break;
    case 'f':
      break;
    default:
      cout << "Opzione errata\n";
    }
  } while (res != 'f');

  return 0;
}
```

3 **albero.h**

```
// -*- C++ -*-
#ifndef ALBERO_H
#define ALBERO_H

struct Nodo {
    int val;
    Nodo *sx;
    Nodo *dx;
};

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, int val);
boolean cerca(const Albero &t, int val);
void stampa(const Albero &t);

#endif
```

3 **albero_.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

void inizializza(Albero &a) {
  a = NULL;
}

boolean vuoto(const Albero &a) {
  return (a == NULL) ? VERO : FALSO;
}

boolean inserisci(Albero &a, int val) {
  // Caso base
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a == NULL) {
       return FALSO;
    }
    a->val = val;
    a->sx = a->dx = NULL;
    return VERO;
  }
  // Caso ricorsivo. Controllo se scendere a sinistra o a destra
  if (val <= a->val) {
    // Scendo a destra
    return inserisci(a->dx, val);
```

```
  } else if (val > a->val) {
    // Scendo a sinistra
    return inserisci(a->sx, val);
  }
}

boolean cerca(const Albero &a, int val) {
  if (vuoto(a) == VERO) {
    return FALSO;
  } else if (val == a->val) {
    // Trovato
    return VERO;
  } else if (val < a->val) {
    // Scendo a destra
    return cerca(a->dx, val);
  } else {
    // Scendo a sinistra
    return cerca(a->sx, val);
  }
}

void stampa(const Albero &a) {
  if (vuoto(a) == FALSO) {
    // Prima stampo gli elementi MINORI di a->val (cioe' quelli a dx)
    stampa(a->dx);
    // Poi stampo a->val
    cout << a-> val << ' ';
    // Poi stampo gli elementi MAGGIORI (cioe' quelli a sx)
    stampa(a->sx);
  }
}
```

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);
}
```