Primo Appello di Programmazione I

09 Gennaio 2013
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 nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file di testo e un intero `N`, copi i primi `N` caratteri del primo file nel secondo file **sostituendo ogni lettera dell'alfabeto con la lettera che si trova 3 posizioni dopo nell'alfabeto** (e.g. la lettera `a` è sostituita con la lettera `d`, `b` con `e`, `c` con `f`, ...). Si noti che:

- le ultime lettere dell'alfabeto sono sostituite con le prime (e.g. la lettera `z` è sostituita con la lettera `c`);
- le lettere maiuscole sono sostituite con lettere maiuscole così come le lettere minuscole sono sostituite con lettere minuscole (e.g. le seguenti sostituzioni non sono valide: `A` con `d` ed `a` con `D`);
- i caratteri che non sono lettere dell'alfabeto (come numeri e simboli) non vengono modificati.

Se ad esempio l'eseguibile è `a.out` ed il file `input` ha il seguente contenuto:

```
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.
```

allora il comando:

```
./a.out input output 120
```

genera un file chiamato `output` con il seguente contenuto:

```
Ilodvwurffd ghooh sduroh:
idwhyl dydqwl! fkl qh yxroh?
Gl sduroh kr od whvwd slhqd,
frq ghqwur od "oxqd" h od "edohqd
```

NOTA 1: Il programma deve terminare con un messaggio appropriato qualora il numero di argomenti sulla linea di comando non sia corretto.

NOTA 2: Il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). È vietato l'uso di tabelle o di `if` o `switch-case`, uno per ogni carattere.

NOTA 3: È ammesso l'uso della funzione `atoi()` (della libreria `cstdlib`) per convertire una stringa in intero.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

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

  fstream my_in, my_out;
  int count, maxnum;
  char c;

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile> <targetfile> <num>\n";
    exit(0);
  }

  maxnum = atoi(argv[3]);
  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  count = 0;
  while(my_in.get(c) && count < maxnum){
     if ( (c>='A' && c<='W') || (c>='a' && c<='w')) {
        c+=3;
     } else if ((c>='X' && c<='Z') || (c>='x' && c<='z')) {
        c-=23;
     }
     count++;
     my_out << c;
  }

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

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file di testo e un intero `N`, copi i primi `N` caratteri del primo file nel secondo file **sostituendo ogni lettera dell'alfabeto con la lettera che si trova 5 posizioni prima nell'alfabeto** (e.g. la lettera `f` è sostituita con la lettera `a`, `g` con `b`, `h` con `c`, ...). Si noti che:

- le prime lettere dell'alfabeto sono sostituite con le ultime (e.g. la lettera `a` è sostituita con la lettera `v`);
- le lettere maiuscole sono sostituite con lettere maiuscole così come le lettere minuscole sono sostituite con lettere minuscole (e.g. le seguenti sostituzioni non sono valide: `F` con `a` ed `f` con `A`);
- i caratteri che non sono lettere dell'alfabeto (come numeri e simboli) non vengono modificati.

Se ad esempio l'eseguibile è `a.out` ed il file `input` ha il seguente contenuto:

```
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.
```

allora il comando:

```
./a.out input output 100
```

genera un file chiamato `output` con il seguente contenuto:

```
Adgvnomjxxv yzggz kvmjgz:
avozqd vqviod! xcd iz qpjgz?
Yd kvmjgz cj gv oznov kdziv,
xji yziomj gv
```

NOTA 1: Il programma deve terminare con un messaggio appropriato qualora il numero di argomenti sulla linea di comando non sia corretto.

NOTA 2: Il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). È vietato l'uso di tabelle o di `if` o `switch-case`, uno per ogni carattere.

NOTA 3: È ammesso l'uso della funzione `atoi()` (della libreria `cstdlib`) per convertire una stringa in intero.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

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

  fstream my_in, my_out;
  int count, maxnum;
  char c;

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile> <targetfile> <num>\n";
    exit(0);
  }

  maxnum = atoi(argv[3]);
  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  count = 0;
  while(my_in.get(c) && count < maxnum){
     if ( (c>='F' && c<='Z') || (c>='f' && c<='z')) {
        c-=5;
     } else if ((c>='A' && c<='E') || (c>='a' && c<='e')) {
        c+=21;
     }
     count++;
     my_out << c;
  }

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

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file di testo e un intero `N`, copi i primi `N` caratteri del primo file nel secondo file **sostituendo ogni lettera minuscola dell'alfabeto con la corrispondente lettera maiuscola** (e.g. la lettera `a` è sostituita con la lettera `A`). Si noti che i caratteri che non sono lettere dell'alfabeto (come numeri e simboli) non vengono modificati.

Se ad esempio l'eseguibile è `a.out` ed il file `input` ha il seguente contenuto:

```
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.
```

allora il comando:

```
./a.out input output 150
```

genera un file chiamato `output` con il seguente contenuto:

```
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 AMI
```

NOTA 1: Il programma deve terminare con un messaggio appropriato qualora il numero di argomenti sulla linea di comando non sia corretto.

NOTA 2: Il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). È vietato l'uso di tabelle o di `if` o `switch-case`, uno per ogni carattere.

NOTA 3: È ammesso l'uso della funzione `atoi()` (della libreria `cstdlib`) per convertire una stringa in intero.

NOTA 4: **Non è ammesso** l'uso delle funzioni di libreria `toupper()` e `tolower()` per convertire un carattere minuscolo in un carattere maiuscolo e viceversa.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

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

  fstream my_in, my_out;
  int count, maxnum;
  char c;

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile> <targetfile> <num>\n";
    exit(0);
  }

  maxnum = atoi(argv[3]);
  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  count = 0;
  while(my_in.get(c) && count < maxnum){
     if (c>='a' && c<='z') {
        c-='a'-'A';
     }
     count++;
     my_out << c;
  }

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

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file di testo e un intero `N`, copi i primi `N` caratteri del primo file nel secondo file **sostituendo ogni lettera maiuscola dell'alfabeto con la corrispondente lettera minuscola** (e.g. la lettera `D` è sostituita con la lettera `d`). Si noti che i caratteri che non sono lettere dell'alfabeto (come numeri e simboli) non vengono modificati.

Se ad esempio l'eseguibile è `a.out` ed il file `input` ha il seguente contenuto:

```
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.
```

allora il comando:

```
./a.out input output 130
```

genera un file chiamato `output` con il seguente contenuto:

```
filastrocca delle parole:
fatevi avanti! chi ne vuole?
di parole ho la testa piena,
con dentro la "luna" e la "balena".
ci son
```

NOTA 1: Il programma deve terminare con un messaggio appropriato qualora il numero di argomenti sulla linea di comando non sia corretto.

NOTA 2: Il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). È vietato l'uso di tabelle o di `if` o `switch-case`, uno per ogni carattere.

NOTA 3: È ammesso l'uso della funzione `atoi()` (della libreria `cstdlib`) per convertire una stringa in intero.

NOTA 4: **Non è ammesso** l'uso delle funzioni di libreria `toupper()` e `tolower()` per convertire un carattere minuscolo in un carattere maiuscolo e viceversa.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

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

  fstream my_in, my_out;
  int count, maxnum;
  char c;

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile> <targetfile> <num>\n";
    exit(0);
  }

  maxnum = atoi(argv[3]);
  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  count = 0;
  while(my_in.get(c) && count < maxnum){
     if (c>='A' && c<='Z') {
        c+='a'-'A';
     }
     count++;
     my_out << c;
  }

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

2 Il file `esercizio2.cc` contiene un programma che manipola matrici di interi, statiche e dinamiche (come matrici dinamiche si intendono array dinamici di array dinamici, come visto a lezione).

Si scrivano le seguenti funzioni:

(a) la funzione `alloca_matrice` che, presi come parametri il numero di righe e il numero di colonne della matrice, restituisca una matrice dinamica di interi allocando dinamicamente la memoria necessaria;

(b) la procedura `dealloca_matrice` che liberi correttamente tutta la memoria allocata dalla matrice;

(c) la procedura `seleziona` che presi come parametri la matrice originale in forma statica, le coordinate del primo elemento e le dimensioni di una sottomatrice (righe e colonne), ritorni tale sottomatrice in forma dinamica.

La funzione `stampa_matrice` si occupa di stampare a video gli elementi della matrice ed è dichiarata nel file `stampamatrice.h`, che viene fornito assieme al file oggetto `stampamatrice.o`. Il programma `esercizio2.cc` dovrà essere quindi compilato utilizzando tale file.

Se ad esempio la matrice originale fosse costituita da `4` righe e `4` colonne

```
1 2 3 4
4 5 6 7
7 8 9 0
8 9 0 1
```

e i valori 2, 1, 2 e 3 come parametri di ingresso, il programma dovrà stampare a video:

```
8 9 0
9 0 1
```

e infine deallocherà la matrice allocata.

NOTE: Si supponga che sia responsabilità dell'utente inserire valori tali che la sottomatrice sia interamente contenuta nella matrice data.

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

2 **esercizio2.cc**

```
#include <iostream>
#include <fstream>
#include "stampamatrice.h"

using namespace std;

const int NUMERO_RIGHE = 6;
const int NUMERO_COLONNE = 5;

int **seleziona(int matrice[][NUMERO_COLONNE], int riga, int colonna, int righe, int colonne);
int **alloca_matrice(int righe, int colonne);
void dealloca_matrice(int **matrice, int righe);

int main ()
{
int matrice[][NUMERO_COLONNE] = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10},
{11, 12, 13, 14, 15}, {16, 17, 18, 19, 20}, {21, 22, 23, 24, 25}, {26, 27, 28, 29, 30}};

int n_riga, n_colonna, n_righe, n_colonne;

// Posizione del primo elemento della selezione
  cout << "Posizione primo elemento della selezione; riga e colonna: ";
  cin >> n_riga >> n_colonna;
// Dimensione della selezione
  cout << "Dimensione della selezione; righe e colonne: ";
  cin >> n_righe >> n_colonne;

  // Selezione della sotto-matrice
  int** selezione = seleziona(matrice, n_riga, n_colonna, n_righe, n_colonne);

  cout << "Matrice di output: " << endl;
  stampa_matrice(selezione, n_righe, n_colonne);

  // Deallocazione della matrice
  dealloca_matrice(selezione, n_righe);

  return 0;
}

int** seleziona(int matrice[NUMERO_RIGHE][NUMERO_COLONNE], int riga, int colonna, int righe, in
  int** selezione = alloca_matrice(righe, colonne);
  for(int i = 0; i<righe; i++) {
for(int j = 0; j<colonne; j++) {
selezione[i][j] = matrice[riga+i][colonna+j];
}
}
return selezione;
}

int **alloca_matrice(int righe, int colonne) {
  int **matrice = new int* [righe];
  for(int i = 0; i<righe; i++) {
```

```
    matrice[i] = new int[colonne];
  }
  return matrice;
}

void dealloca_matrice(int **matrice, int righe) {
  for(int i = 0; i<righe; i++) {
    delete [] matrice[i];
  }
  delete [] matrice;
}
```

2 Il file `esercizio2.cc` contiene un programma che manipola matrici di caratteri, statiche e dinamiche (come matrici dinamiche si intendono array dinamici di array dinamici, come visto a lezione).

Si scrivano le seguenti funzioni:

(a) la funzione `alloca_matrice` che, presi come parametri il numero di righe e il numero di colonne della matrice, restituisca una matrice dinamica di caratteri allocando dinamicamente la memoria necessaria;

(b) la procedura `dealloca_matrice` che liberi correttamente tutta la memoria allocata dalla matrice;

(c) la procedura `seleziona` che presi come parametri la matrice originale in forma statica, le coordinate del primo elemento e le dimensioni di una sottomatrice (righe e colonne), ritorni tale sottomatrice in forma dinamica.

La funzione `stampa_matrice` si occupa di stampare a video gli elementi della matrice ed è dichiarata nel file `stampamatrice.h`, che viene fornito assieme al file oggetto `stampamatrice.o`. Il programma `esercizio2.cc` dovrà essere quindi compilato utilizzando tale file.

Se ad esempio la matrice originale fosse costituita da `4` righe e `4` colonne

```
a b c d
e f g h
* / + -
j k l m
```

e i valori 2, 1, 2 e 3 come parametri di ingresso, il programma dovrà stampare a video:

```
/ + -
k l m
```

e infine deallocherà la matrice allocata.

NOTE: Si supponga che sia responsabilità dell'utente inserire valori tali che la sottomatrice sia interamente contenuta nella matrice data.

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

2 **esercizio2.cc**

```
#include <iostream>
#include <fstream>
#include "stampamatrice.h"

using namespace std;

const int NUMERO_RIGHE = 6;
const int NUMERO_COLONNE = 5;

bool controlla(int riga, int colonna, int righe, int colonne);
char **seleziona(char matrice[][NUMERO_COLONNE], int riga, int colonna, int righe, int colonne)
char **alloca_matrice(int righe, int colonne);
void dealloca_matrice(char **matrice, int righe);

int main ()
{
char matrice[][NUMERO_COLONNE] = {{'a', 'b', 'c', 'd', 'e'}, {'f', 'g', 'h', 'i', 'j'},
{'k', 'l', 'm', 'n', 'o'}, {'p', 'q', 'r', 's', 't'}, {'u', 'v', 'w', 'x', 'y'}, {'z', '+', '-'

int n_riga, n_colonna, n_righe, n_colonne;

// Posizione del primo elemento della selezione
  cout << "Posizione primo elemento della selezione; riga e colonna: ";
  cin >> n_riga >> n_colonna;
// Dimensione della selezione
  cout << "Dimensione della selezione; righe e colonne: ";
  cin >> n_righe >> n_colonne;

  // Selezione della sotto-matrice
  char** selezione = seleziona(matrice, n_riga, n_colonna, n_righe, n_colonne);

  cout << "Matrice di output: " << endl;
  stampa_matrice(selezione, n_righe, n_colonne);

  // Deallocazione della matrice
  dealloca_matrice(selezione, n_righe);

  return 0;
}

char** seleziona(char matrice[NUMERO_RIGHE][NUMERO_COLONNE], int riga, int colonna, int righe,
  char** selezione = alloca_matrice(righe, colonne);
  for(int i = 0; i<righe; i++) {
for(int j = 0; j<colonne; j++) {
selezione[i][j] = matrice[riga+i][colonna+j];
}
}
return selezione;
}

char **alloca_matrice(int righe, int colonne) {
  char **matrice = new char* [righe];
```

```
  for(int i = 0; i<righe; i++) {
    matrice[i] = new char[colonne];
  }
  return matrice;
}

void dealloca_matrice(char **matrice, int righe) {
  for(int i = 0; i<righe; i++) {
    delete [] matrice[i];
  }
  delete [] matrice;
}
```

2 Il file `esercizio2.cc` contiene un programma che manipola matrici di interi, statiche e dinamiche (come matrici dinamiche si intendono array dinamici di array dinamici, come visto a lezione).

Si scrivano le seguenti funzioni:

(a) la funzione `alloca_matrice` che, presi come parametri il numero di righe e il numero di colonne della matrice, restituisca una matrice dinamica di interi allocando dinamicamente la memoria necessaria;

(b) la procedura `dealloca_matrice` che liberi correttamente tutta la memoria allocata dalla matrice;

(c) la procedura `seleziona` che presi come parametri la matrice originale in forma statica e le coordinate di due elementi, che corrispondono rispettivamente agli angoli "in alto a sinistra" e "in basso a destra" di una sottomatrice cosí individuata, ritorni tale sottomatrice in forma dinamica.

La funzione `stampa_matrice` si occupa di stampare a video gli elementi della matrice ed è dichiarata nel file `stampamatrice.h`, che viene fornito assieme al file oggetto `stampamatrice.o`. Il programma `esercizio2.cc` dovrà essere quindi compilato utilizzando tale file.

Se ad esempio la matrice originale fosse costituita da `4` righe e `4` colonne

```
1 2 3 4
4 5 6 7
7 8 9 0
8 9 0 1
```

e i valori 2, 1, 3 e 3 come parametri di ingresso, il programma dovrà stampare a video:

```
8 9 0
9 0 1
```

e infine deallocherà la matrice allocata.

NOTE: Si supponga che sia responsabilità dell'utente inserire valori tali che la sottomatrice sia interamente contenuta nella matrice data.

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

2 **esercizio2.cc**

```
#include <iostream>
#include <fstream>
#include "stampamatrice.h"

using namespace std;

const int NUMERO_RIGHE = 6;
const int NUMERO_COLONNE = 5;

int **seleziona(int matrice[][NUMERO_COLONNE], int a, int b, int c, int d);
int **alloca_matrice(int righe, int colonne);
void dealloca_matrice(int **matrice, int a, int c);

int main ()
{
int matrice[][NUMERO_COLONNE] = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10},
{11, 12, 13, 14, 15}, {16, 17, 18, 19, 20}, {21, 22, 23, 24, 25}, {26, 27, 28, 29, 30}};

int n_riga_1, n_colonna_1, n_riga_2, n_colonna_2;

// Posizione del primo elemento della selezione
  cout << "Posizione primo elemento della selezione; riga e colonna: ";
  cin >> n_riga_1 >> n_colonna_1;
// Posizione del secondo elemento della selezione
  cout << "Posizione secondo elemento della selezione; riga e colonna: ";

  cin >> n_riga_2 >> n_colonna_2;

  // Selezione della sotto-matrice
  int** selezione = seleziona(matrice, n_riga_1, n_colonna_1, n_riga_2, n_colonna_2);

  cout << "Matrice di output: " << endl;
  stampa_matrice(selezione, n_riga_1, n_colonna_1, n_riga_2, n_colonna_2);

  // Deallocazione delle matrici
  dealloca_matrice(selezione, n_riga_1, n_riga_2);

  return 0;
}

int** seleziona(int matrice[NUMERO_RIGHE][NUMERO_COLONNE], int a, int b, int c, int d) {
  int** selezione = alloca_matrice(c - a + 1, d - b + 1);
  for(int i = 0; i<(c - a + 1); i++) {
for(int j = 0; j<(d - b + 1); j++) {
selezione[i][j] = matrice[a+i][b+j];
}
}
return selezione;
}

int **alloca_matrice(int righe, int colonne) {
  int **matrice = new int* [righe];
```

```
  for(int i = 0; i<righe; i++) {
    matrice[i] = new int[colonne];
  }
  return matrice;
}

void dealloca_matrice(int **matrice, int riga_1, int riga_2) {
  for(int i = 0; i<(riga_2 - riga_1 + 1); i++) {
    delete [] matrice[i];
  }
  delete [] matrice;
}
```

2 Il file `esercizio2.cc` contiene un programma che manipola matrici di caratteri, statiche e dinamiche (come matrici dinamiche si intendono array dinamici di array dinamici, come visto a lezione).

Si scrivano le seguenti funzioni:

(a) la funzione `alloca_matrice` che, presi come parametri il numero di righe e il numero di colonne della matrice, restituisca una matrice dinamica di caratteri allocando dinamicamente la memoria necessaria;

(b) la procedura `dealloca_matrice` che liberi correttamente tutta la memoria allocata dalla matrice;

(c) la procedura `seleziona` che presi come parametri la matrice originale in forma statica e le coordinate di due elementi, che corrispondono rispettivamente agli angoli "in alto a sinistra" e "in basso a destra" di una sottomatrice così individuata, ritorni tale sottomatrice in forma dinamica.

La funzione `stampa_matrice` si occupa di stampare a video gli elementi della matrice ed è dichiarata nel file `stampamatrice.h`, che viene fornito assieme al file oggetto `stampamatrice.o`. Il programma `esercizio2.cc` dovrà essere quindi compilato utilizzando tale file.

Se ad esempio la matrice originale fosse costituita da `4` righe e `4` colonne

```
a b c d
e f g h
* / + -
j k l m
```

e i valori 2, 1, 3 e 3 come parametri di ingresso, il programma dovrà stampare a video:

```
/ + -
k l m
```

e infine deallocherà la matrice allocata.

NOTE: Si supponga che sia responsabilità dell'utente inserire valori tali che la sottomatrice sia interamente contenuta nella matrice data.

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

2 **esercizio2.cc**

```
#include <iostream>
#include <fstream>
#include "stampamatrice.h"

using namespace std;

const int NUMERO_RIGHE = 6;
const int NUMERO_COLONNE = 5;

char **seleziona(char matrice[][NUMERO_COLONNE], int a, int b, int c, int d);
char **alloca_matrice(int righe, int colonne);
void dealloca_matrice(char **matrice, int a, int c);

int main () {
  char matrice[][NUMERO_COLONNE] = {{'a', 'b', 'c', 'd', 'e'}, {'f', 'g', 'h', 'i', 'j'},
{'k', 'l', 'm', 'n', 'o'}, {'p', 'q', 'r', 's', 't'}, {'u', 'v', 'w', 'x', 'y'}, {'z', '+', '-'

int n_riga_1, n_colonna_1, n_riga_2, n_colonna_2;

// Posizione del primo elemento della selezione
  cout << "Posizione primo elemento della selezione; riga e colonna: ";
  cin >> n_riga_1 >> n_colonna_1;
// Posizione del secondo elemento della selezione
  cout << "Posizione secondo elemento della selezione; riga e colonna: ";
  cin >> n_riga_2 >> n_colonna_2;

  // Selezione della sotto-matrice
  char** selezione = seleziona(matrice, n_riga_1, n_colonna_1, n_riga_2, n_colonna_2);

  cout << "Matrice di output: " << endl;
  stampa_matrice(selezione, n_riga_1, n_colonna_1, n_riga_2, n_colonna_2);

  // Deallocazione delle matrici
  dealloca_matrice(selezione, n_riga_1, n_riga_2);

  return 0;
}

char** seleziona(char matrice[NUMERO_RIGHE][NUMERO_COLONNE], int a, int b, int c, int d) {
  char** selezione = alloca_matrice(c - a + 1, d - b + 1);
  for(int i = 0; i<(c - a + 1); i++) {
for(int j = 0; j<(d - b + 1); j++) {
selezione[i][j] = matrice[a+i][b+j];
}
}
return selezione;
}

char **alloca_matrice(int righe, int colonne) {
  char **matrice = new char* [righe];
  for(int i = 0; i<righe; i++) {
    matrice[i] = new char[colonne];
```

```
  }
  return matrice;
}

void dealloca_matrice(char **matrice, int a, int c) {
  for(int i = 0; i<(c - a + 1); i++) {
    delete [] matrice[i];
  }
  delete [] matrice;
}
```

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

- `init` inizializzi l'albero;
- `empty` controlli se l'albero è vuoto, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `insert` inserisca l'elemento passato come parametro nell'albero, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` 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:

  ```
  e  a  u  a
  ```

  deve produrre il seguente albero:

  ```
        e
     a     u
  a
  ```

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

  ```
  a  a  e  u
  ```

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

3 albero_main.cc

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

int main()
{
    char res;
    Tree tree;
    char val;
    init(tree);
    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 (insert(tree, val) == FALSE) {
                cout << "Valore gia' presente!" << endl;
            }
            break;
        case 'r':
            cout << "Valore: ";
            cin >> val;
            if (search(tree, val) == TRUE) {
                cout << "Valore presente: " << val << endl;
            } else {
                cout << "Valore non presente" << endl;
            }
            break;
        case 's':
            if (empty(tree) == TRUE) {
                cout << "Albero vuoto!" << endl;
            } else {
                print(tree);
            }
            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 Node {
    char val;
    Node *left;
    Node *right;
};

typedef Node * Tree;

enum boolean { FALSE, TRUE };

void init(Tree &t);
boolean empty(const Tree &t);
boolean insert(Tree &t, char val);
boolean search(const Tree &t, char val);
void print(const Tree &t);

#endif
```

3 **soluzione_A31.cc**

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

void init(Tree &t)
{
    t = NULL;
}


boolean empty(const Tree &t)
{
    return (t == NULL) ? TRUE : FALSE;
}


boolean insert(Tree &t, char val)
{
    // caso base
    if (empty(t) == TRUE) {
        t = new (nothrow) Node;
        if (t == NULL) {
           return FALSE;
        }
        t->val = val;
        t->left = t->right = NULL;
        return TRUE;
    }
```

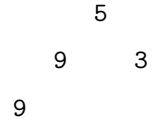
```
    // caso ricorsivo. Controllo se scendere a sinistra o a destra
    if (val <= t->val) {
        // scendo a sinistra
        return insert(t->left, val);
    } else if (val > t->val) {
        // scendo a destra
        return insert(t->right, val);
    }
}



boolean search(const Tree &t, char val)
{
    if (empty(t) == TRUE) {
        return FALSE;
    } else if (val == t->val) {
        return TRUE;
    } else if (val < t->val) {
        // scendo a sinistra
        return search(t->left, val);
    } else {
        // scendo a destra
        return search(t->right, val);
    }
}


void print(const Tree &t)
{
    if (empty(t) == FALSE) {
        // prima stampo gli elementi minori di t->val (cioe' quelli a sx)
        print(t->left);
        // poi stampo t->val
        cout << t-> val << ' ';
        // poi stampo gli elementi maggiori (cioe' quelli a dx)
        print(t->right);
    }
}
```

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

- `init` inizializzi l'albero;
- `empty` controlli se l'albero è vuoto, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `insert` inserisca l'elemento passato come parametro nell'albero, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` 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:

  ```
  5 9 3 9
  ```

  deve produrre il seguente albero:

- `search` cerchi nell'albero l'elemento passato in input, resituendo `TRUE` se l'elemento è presente, e `FALSE` altrimenti;
- `print` stampi a video il contenuto dell'albero, in ordine **decrescente**. Esempio: l'albero qui sopra deve essere stampato come:

  ```
  9 9 5 3
  ```

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

3 albero_main.cc

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

int main()
{
    char res;
    Tree tree;
    int val;
    init(tree);
    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 (insert(tree, val) == FALSE) {
                cout << "Valore gia' presente!" << endl;
            }
            break;
        case 'r':
            cout << "Valore: ";
            cin >> val;
            if (search(tree, val) == TRUE) {
                cout << "Valore presente: " << val << endl;
            } else {
                cout << "Valore non presente" << endl;
            }
            break;
        case 's':
            if (empty(tree) == TRUE) {
                cout << "Albero vuoto!" << endl;
            } else {
                print(tree);
            }
            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 Node {
    int val;
    Node *left;
    Node *right;
};

typedef Node * Tree;

enum boolean { FALSE, TRUE };

void init(Tree &t);
boolean empty(const Tree &t);
boolean insert(Tree &t, int val);
boolean search(const Tree &t, int val);
void print(const Tree &t);

#endif
```

3 **soluzione_A32.cc**

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

void init(Tree &t)
{
    t = NULL;
}


boolean empty(const Tree &t)
{
    return (t == NULL) ? TRUE : FALSE;
}


boolean insert(Tree &t, int val)
{
    // caso base
    if (empty(t) == TRUE) {
        t = new (nothrow) Node;
        if (t == NULL) {
           return FALSE;
        }
        t->val = val;
        t->left = t->right = NULL;
        return TRUE;
    }
```

```
    // caso ricorsivo. Controllo se scendere a sinistra o a destra
    if (val >= t->val) {
        // scendo a sinistra
        return insert(t->left, val);
    } else if (val < t->val) {
        // scendo a destra
        return insert(t->right, val);
    }
}



boolean search(const Tree &t, int val)
{
    if (empty(t) == TRUE) {
        return FALSE;
    } else if (val == t->val) {
        return TRUE;
    } else if (val > t->val) {
        // scendo a sinistra
        return search(t->left, val);
    } else {
        // scendo a destra
        return search(t->right, val);
    }
}


void print(const Tree &t)
{
    if (empty(t) == FALSE) {
        // prima stampo gli elementi minori di t->val (cioe' quelli a sx)
        print(t->left);
        // poi stampo t->val
        cout << t-> val << ' ';
        // poi stampo gli elementi maggiori (cioe' quelli a dx)
        print(t->right);
    }
}
```

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

- `init` inizializzi l'albero;
- `empty` controlli se l'albero è vuoto, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `insert` inserisca l'elemento passato come parametro nell'albero, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` 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:

  ```
  6.5  2.1  8.8  2.1
  ```

  deve produrre il seguente albero:

  ```
          6.5
    8.8         2.1
                      2.1
  ```

- `search` cerchi nell'albero l'elemento passato in input, resituendo `TRUE` se l'elemento è presente, e `FALSE` altrimenti;
- `print` stampi a video il contenuto dell'albero, in ordine **decrescente**. Esempio: l'albero qui sopra deve essere stampato come:

  ```
  8.8  6.5  2.1  2.1
  ```

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

3 **albero_main.cc**

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

int main()
{
    char res;
    Tree tree;
    double val;
    init(tree);
    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 (insert(tree, val) == FALSE) {
                cout << "Valore gia' presente!" << endl;
            }
            break;
        case 'r':
            cout << "Valore: ";
            cin >> val;
            if (search(tree, val) == TRUE) {
                cout << "Valore presente: " << val << endl;
            } else {
                cout << "Valore non presente" << endl;
            }
            break;
        case 's':
            if (empty(tree) == TRUE) {
                cout << "Albero vuoto!" << endl;
            } else {
                print(tree);
            }
            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 Node {
    double val;
    Node *left;
    Node *right;
};

typedef Node * Tree;

enum boolean { FALSE, TRUE };

void init(Tree &t);
boolean empty(const Tree &t);
boolean insert(Tree &t, double val);
boolean search(const Tree &t, double val);
void print(const Tree &t);

#endif
```

3 **soluzione_A33.cc**

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

void init(Tree &t)
{
    t = NULL;
}


boolean empty(const Tree &t)
{
    return (t == NULL) ? TRUE : FALSE;
}


boolean insert(Tree &t, double val)
{
    // caso base
    if (empty(t) == TRUE) {
        t = new (nothrow) Node;
        if (t == NULL) {
           return FALSE;
        }
        t->val = val;
        t->left = t->right = NULL;
        return TRUE;
    }
```

```
    // caso ricorsivo. Controllo se scendere a sinistra o a destra
    if (val > t->val) {
        // scendo a sinistra
        return insert(t->left, val);
    } else if (val <= t->val) {
        // scendo a destra
        return insert(t->right, val);
    }
}



boolean search(const Tree &t, double val)
{
    if (empty(t) == TRUE) {
        return FALSE;
    } else if (val == t->val) {
        return TRUE;
    } else if (val > t->val) {
        // scendo a sinistra
        return search(t->left, val);
    } else {
        // scendo a destra
        return search(t->right, val);
    }
}


void print(const Tree &t)
{
    if (empty(t) == FALSE) {
        // prima stampo gli elementi minori di t->val (cioe' quelli a sx)
        print(t->left);
        // poi stampo t->val
        cout << t-> val << ' ';
        // poi stampo gli elementi maggiori (cioe' quelli a dx)
        print(t->right);
    }
}
```

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

- `init` inizializzi l'albero;
- `empty` controlli se l'albero è vuoto, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `insert` inserisca l'elemento passato come parametro nell'albero, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` 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:

  ```
  5.1  9.3  3.5  9.3
  ```

  deve produrre il seguente albero:

  ```
          5.1
    3.5         9.3
                      9.3
  ```

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

  ```
  3.5  5.1  9.3  9.3
  ```

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

3 **albero_main.cc**

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

int main()
{
    char res;
    Tree tree;
    float val;
    init(tree);
    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 (insert(tree, val) == FALSE) {
                cout << "Valore gia' presente!" << endl;
            }
            break;
        case 'r':
            cout << "Valore: ";
            cin >> val;
            if (search(tree, val) == TRUE) {
                cout << "Valore presente: " << val << endl;
            } else {
                cout << "Valore non presente" << endl;
            }
            break;
        case 's':
            if (empty(tree) == TRUE) {
                cout << "Albero vuoto!" << endl;
            } else {
                print(tree);
            }
            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 Node {
    float val;
    Node *left;
    Node *right;
};

typedef Node * Tree;

enum boolean { FALSE, TRUE };

void init(Tree &t);
boolean empty(const Tree &t);
boolean insert(Tree &t, float val);
boolean search(const Tree &t, float val);
void print(const Tree &t);

#endif
```

3 **soluzione_A34.cc**

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

void init(Tree &t)
{
    t = NULL;
}


boolean empty(const Tree &t)
{
    return (t == NULL) ? TRUE : FALSE;
}


boolean insert(Tree &t, float val)
{
    // caso base
    if (empty(t) == TRUE) {
        t = new (nothrow) Node;
        if (t == NULL) {
           return FALSE;
        }
        t->val = val;
        t->left = t->right = NULL;
        return TRUE;
    }
```

```
    // caso ricorsivo. Controllo se scendere a sinistra o a destra
    if (val < t->val) {
        // scendo a sinistra
        return insert(t->left, val);
    } else if (val >= t->val) {
        // scendo a destra
        return insert(t->right, val);
    }
}



boolean search(const Tree &t, float val)
{
    if (empty(t) == TRUE) {
        return FALSE;
    } else if (val == t->val) {
        return TRUE;
    } else if (val < t->val) {
        // scendo a sinistra
        return search(t->left, val);
    } else {
        // scendo a destra
        return search(t->right, val);
    }
}


void print(const Tree &t)
{
    if (empty(t) == FALSE) {
        // prima stampo gli elementi minori di t->val (cioe' quelli a sx)
        print(t->left);
        // poi stampo t->val
        cout << t-> val << ' ';
        // poi stampo gli elementi maggiori (cioe' quelli a dx)
        print(t->right);
    }
}
```

4 Dati i seguenti file:

- `persona.h` e `persona.o` contenenti le strutture dati e le funzioni per la gestione di una persona (e.g. creazione e stampa di una persona);
- `tree.h` e `tree.o` contenenti le strutture dati e le funzioni per la gestione di un albero binario di ricerca di puntatori a persona ordinato in maniera crescente in base al cognome;
- `queue.h` e `queue.o` contenenti le strutture dati e le funzioni per la gestione di una coda di puntatori a persona;
- `esercizio4.cc` contenente il main del programma che implementa un menu per la gestione un'agenda di persone.

Scrivere all'interno del file `esercizio4.cc` la dichiarazione e la definizione della funzione `elenca_tutte` che, presi come parametri un'agenda di persone ed una stringa `s`, ritorni una coda (ordinata) di persone il cui cognome inizia per `s`.

NOTA 1: La funzione `elenca_tutte` può essere iterativa o ricorsiva e può far uso di funzioni ausiliarie.

NOTA 2: È ammesso l'uso delle funzioni della librearia `cstring`.

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

4 **soluzione_A41.cc**

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

#include "persona.h"
#include "tree.h"
#include "queue.h"

// Inserisci qui sotto la dichiarazione della funzione elenca_tutte

queue elenca_tutte(tree agenda, char* cognome);

int main ()
{

  persona * p;
  tree  agenda;
  char cognome[MAX_DIM];
  char indirizzo[MAX_DIM];
  char res;
  tree  tmp;
  queue q;

  init(agenda);

    do {
    cout << "\nOperazioni possibili:\n"
         << "Inserimento (i)\n"
         << "Ricerca per cognome (r)\n"
         << "Stampa ordinata (s)\n"
         << "Modifica indirizzo (m)\n"
         << "Elenca tutte (e)\n"
         << "Fine (f)\n";
    cin >> res;
    switch (res) {
    case 'i':
      p = leggi_persona();
      agenda = inserisci(agenda,p);
      break;
    case 'r':
      cout << "Cognome? : ";
      cin >> cognome;
      tmp=cerca(agenda,cognome);
      if (tmp!=NULL)
        stampa_persona(tmp->p);
      else
        cout << "Valore non trovato!\n";
      break;
    case 's':
      cout << "Stampa ordinata:\n";
      stampa_ordinata(agenda);
      break;
    case 'm':
```

```
      cout << "Cognome? : ";
      cin >> cognome;
      cout << "Nuovo indirizzo? : ";
      cin >> indirizzo;
      tmp=modifica_indirizzo(agenda,cognome,indirizzo);
      if (tmp==NULL)
        cout << "Valore non trovato!\n";
      break;
    case 'e':
      cout << "Cognome o sottostringa iniziale? : ";
      cin >> cognome;
      q=elenca_tutte(agenda,cognome);
      if (emptyp(q))
         cout << "Valore non trovato!\n";
      else
         cout << "Elenco persone :\n";
         print(q);
      break;
    case 'f':
      break;
    default:
      cout << "Optione errata\n";
    }
  } while (res != 'f');
}

// Inserisci qui sotto la definizione della funzione elenca_tutte

void elenca_tutte_ric(tree t, char* s, queue &q) {
  if (t==NULL) {
    return;
  }

  char* p = strstr(t->p->cognome, s);

  if (strcmp(s,t->p->cognome)<0) {
     elenca_tutte_ric(t->left,s,q);
  }

  if (p == t->p->cognome) {
     enqueue(t->p, q);
  }

  if (strcmp(s,t->p->cognome)>=0) {
     elenca_tutte_ric(t->right,s,q);
  }
}

queue elenca_tutte(tree agenda, char* cognome) {
  queue q;
  init(q);
  elenca_tutte_ric(agenda, cognome, q);
  return q;
}
```