Primo Appello di Programmazione I

10 Gennaio 2018
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 da riga di comando:

(a) il nome di un file di testo in input;
(b) un numero intero "**n**", corrispondente al numero di righe da leggere dal primo file;
(c) il nome di un file di testo in output.

legga le prime **n** righe del primo file, o tutte le righe del file se queste ultime sono meno di **n**, le processi come specificato sotto e poi salvi le righe risultanti **in ordine inverso** nel secondo file. Ogni riga deve essere riscritta come segue: ogni carattere separatore (' ', '\t') deve rimanere invariato, mentre ogni altro carattere deve essere sostituito con il carattere '*****' se si trova nella riga in posizione dispari, con il carattere '+' altrimenti.

Utilizzare la funzione `atoi` della libreria `cstdlib` per leggere il numero di righe dal secondo parametro in ingresso.

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

```
La donzelletta vien dalla campagna,
in sul calar del sole,
col suo fascio dell'erba, e reca in mano
un mazzolin di rose e di viole.
(da "Il sabato del villaggio" di G. Leopardi)
```

allora il comando:

```
./a.out input.txt 4 output.txt
```

crea il file `output.txt` contenente il seguente testo:

```
** +******* ** +*** * ** +*****
*** *** ****** +********* * **** ** ****
** *** ***** *** *****
** ********** **** ***** *********
```

mentre il comando:

```
./a.out input.txt 6 output.txt
```

crea il file `output.txt` contenente il seguente testo:

```
*** *** ****** *** ********* ** ** *********
** +******* ** +*** * ** +*****
*** *** ****** +********* * **** ** ****
** *** ***** *** *****
** ********** **** ***** *********
```

NOTA 1: Si assuma che nessuna riga del file di ingresso sia più lunga di **255** caratteri.

NOTA 2: Il programma non deve prevedere alcun numero massimo di parole o di righe leggibili dal file di input, pena l'annullamento dell'esercizio.

NOTA 3: È ammesso l'utilizzo delle funzioni contenute nelle librerie **cstring**.

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

1 **soluzione_A11.cc**

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

using namespace std;

const int DIMENSIONE_RIGA = 255 + 1;

int main(int argc, char * argv[]) {
  fstream in, out;
  int n_righe;
  char riga[DIMENSIONE_RIGA];
  char** righe;

  // Controllo argomenti passati in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_di_input> <numero_righe> <file_di_output>" << endl;
    exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  in.open(argv[1], ios::in);
  if (in.fail()) {
    cerr << "Il file " << argv[1] << " non esiste o non e' accessibile.\n";
    exit(EXIT_FAILURE);
  }

  // Legge il numero di righe dal parametro a riga di comando
  // (supponiamo sia sempre <= al numero effettivo di righe del file)
  n_righe = atoi(argv[2]);

  // Alloca lo spazio per le righe
  righe = new char* [n_righe];
  // Leggo il file di input, riga per riga
  int r = 0;
  while(in.getline(riga, DIMENSIONE_RIGA) && r < n_righe) {
    int lun = strlen(riga);
    // Alloca lo spazio per questa riga
    righe[r] = new char[lun + 1];
    for(int i = 0; i < lun; i++) {
      if(riga[i] != ' ' && riga[i] != '\t') {
        // Trasformo il carattere secondo l'algoritmo
        righe[r][i] = (i % 2 == 0 ? '*' : '+');
      } else {
        // Aggiungo il separatore appena letto
        righe[r][i] = riga[i];
      }
    }
    // Aggiungo il terminatore di stringa
    righe[r][lun] = '\0';
    // Aggiorno il contatore delle righe lette
```

```
    r++;
  }

  // Chiude lo stream di input
  in.close();

  // Supponiamo che l'apertura del file di output
  // vada sempre a buon fine
  out.open(argv[3], ios::out);

  // Salva le righe al contrario
  for(int i = r - 1; i >= 0; i--) {
    out << righe[i] << endl;
  }

  // Chiude lo stream di output
  out.close();

  return 0;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti da riga di comando:

(a) il nome di un file di testo in input;
(b) un numero intero "**n**", corrispondente al numero di righe da leggere dal primo file;
(c) il nome di un file di testo in output.

legga le prime **n** righe del primo file, o tutte le righe del file se queste ultime sono meno di **n**, le processi come specificato sotto e poi salvi le righe risultanti **in ordine inverso** nel secondo file. Ogni riga deve essere riscritta come segue: ogni carattere separatore (' ', '\t') deve rimanere invariato, mentre ogni altro carattere deve essere sostituito con il carattere '**!**' se si trova in una riga dispari, con il carattere '**?**' se si trova in una riga pari.

Utilizzare la funzione `atoi` della libreria `cstdlib` per leggere il numero di righe dal secondo parametro in ingresso.

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

```
La donzelletta vien dalla campagna,
in sul calar del sole,
col suo fascio dell'erba, e reca in mano
un mazzolin di rose e di viole.
(da "Il sabato del villaggio" di G. Leopardi)
```

allora il comando:

```
./a.out input.txt 4 output.txt
```

crea il file `output.txt` contenente il seguente testo:

```
?? ???????? ?? ???? ? ?? ??????
!!! !!! !!!!!! !!!!!!!!!! ! !!!! !! !!!!
?? ??? ????? ??? ?????
!! !!!!!!!!!! !!!! !!!!! !!!!!!!!!
```

mentre il comando:

```
./a.out input.txt 6 output.txt
```

crea il file `output.txt` contenente il seguente testo:

```
!!! !!! !!!!!! !!! !!!!!!!!!! !! !! !!!!!!!!!
?? ???????? ?? ???? ? ?? ??????
!!! !!! !!!!!! !!!!!!!!!! ! !!!! !! !!!!
?? ??? ????? ??? ?????
!! !!!!!!!!!! !!!! !!!!! !!!!!!!!!
```

NOTA 1: Si assuma che nessuna riga del file di ingresso sia più lunga di **255** caratteri.

NOTA 2: Il programma non deve prevedere alcun numero massimo di parole o di righe leggibili dal file di input, pena l'annullamento dell'esercizio.

NOTA 3: È ammesso l'utilizzo delle funzioni contenute nelle librerie **cstring**.

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

1 **soluzione_A12.cc**

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

using namespace std;

const int DIMENSIONE_RIGA = 255 + 1;

int main(int argc, char * argv[]) {
  fstream in, out;
  int n_righe;
  char riga[DIMENSIONE_RIGA];
  char** righe;

  // Controllo argomenti passati in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_di_input> <numero_righe> <file_di_output>" << endl;
    exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  in.open(argv[1], ios::in);
  if (in.fail()) {
    cerr << "Il file " << argv[1] << " non esiste o non e' accessibile.\n";
    exit(EXIT_FAILURE);
  }

  // Legge il numero di righe dal parametro a riga di comando
  // (supponiamo sia sempre <= al numero effettivo di righe del file)
  n_righe = atoi(argv[2]);

  // Alloca lo spazio per le righe
  righe = new char* [n_righe];
  // Leggo il file di input, riga per riga
  int r = 0;
  while(in.getline(riga, DIMENSIONE_RIGA) && r < n_righe) {
    int lun = strlen(riga);
    // Alloca lo spazio per questa riga
    righe[r] = new char[lun + 1];
    for(int i = 0; i < lun; i++) {
      if(riga[i] != ' ' && riga[i] != '\t') {
        // Trasformo il carattere secondo l'algoritmo
        righe[r][i] = (r % 2 == 0 ? '!' : '?');
      } else {
        // Aggiungo il separatore appena letto
        righe[r][i] = riga[i];
      }
    }
    // Aggiungo il terminatore di stringa
    righe[r][lun] = '\0';
    // Aggiorno il contatore delle righe lette
```

```
    r++;
  }

  // Chiude lo stream di input
  in.close();

  // Supponiamo che l'apertura del file di output
  // vada sempre a buon fine
  out.open(argv[3], ios::out);

  // Salva le righe al contrario
  for(int i = r - 1; i >= 0; i--) {
    out << righe[i] << endl;
  }

  // Chiude lo stream di output
  out.close();

  return 0;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti da riga di comando:

(a) il nome di un file di testo in input;
(b) un numero intero "**n**", corrispondente al numero di righe da leggere dal primo file;
(c) il nome di un file di testo in output.

legga le prime **n** righe del primo file, o tutte le righe del file se queste ultime sono meno di **n**, le processi come specificato sotto e poi salvi le righe risultanti **in ordine inverso** nel secondo file. Ogni riga deve essere riscritta come segue: ogni carattere separatore (' ', '\t') deve essere sostituito con il carattere '**$**' se si trova nella riga in posizione dispari, con il carattere '**%**' se si trova in posizione pari; ogni altro carattere deve rimanere invariato.

Utilizzare la funzione `atoi` della libreria `cstdlib` per leggere il numero di righe dal secondo parametro in ingresso.

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

```
La donzelletta vien dalla campagna,
in sul calar del sole,
col suo fascio dell'erba, e reca in mano
un mazzolin di rose e di viole.
(da "Il sabato del villaggio" di G. Leopardi)
```

allora il comando:

```
./a.out input.txt 4 output.txt
```

crea il file `output.txt` contenente il seguente testo:

```
un$mazzolin%di$rose%e%di$viole.
col%suo%fascio$dell'erba,%e%reca$in%mano
in$sul$calar$del$sole,
La$donzelletta$vien%dalla%campagna,
```

mentre il comando:

```
./a.out input.txt 6 output.txt
```

crea il file `output.txt` contenente il seguente testo:

```
(da%"Il%sabato$del$villaggio"%di$G.%Leopardi)
un$mazzolin%di$rose%e%di$viole.
col%suo%fascio$dell'erba,%e%reca$in%mano
in$sul$calar$del$sole,
La$donzelletta$vien%dalla%campagna,
```

NOTA 1: Si assuma che nessuna riga del file di ingresso sia più lunga di **255** caratteri.

NOTA 2: Il programma non deve prevedere alcun numero massimo di parole o di righe leggibili dal file di input, pena l'annullamento dell'esercizio.

NOTA 3: È ammesso l'utilizzo delle funzioni contenute nelle librerie **cstring**.

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

1 **soluzione_A13.cc**

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

using namespace std;

const int DIMENSIONE_RIGA = 255 + 1;

int main(int argc, char * argv[]) {
  fstream in, out;
  int n_righe;
  char riga[DIMENSIONE_RIGA];
  char** righe;

  // Controllo argomenti passati in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_di_input> <numero_righe> <file_di_output>" << endl;
    exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  in.open(argv[1], ios::in);
  if (in.fail()) {
    cerr << "Il file " << argv[1] << " non esiste o non e' accessibile.\n";
    exit(EXIT_FAILURE);
  }

  // Legge il numero di righe dal parametro a riga di comando
  // (supponiamo sia sempre <= al numero effettivo di righe del file)
  n_righe = atoi(argv[2]);

  // Alloca lo spazio per le righe
  righe = new char* [n_righe];
  // Leggo il file di input, riga per riga
  int r = 0;
  while(in.getline(riga, DIMENSIONE_RIGA) && r < n_righe) {
    int lun = strlen(riga);
    // Alloca lo spazio per questa riga
    righe[r] = new char[lun + 1];
    for(int i = 0; i < lun; i++) {
      if(riga[i] != ' ' && riga[i] != '\t') {
        // Ricopio il carattere appena letto
        righe[r][i] = riga[i];
      } else {
        // Trasformo il carattere secondo l'algoritmo
        righe[r][i] = (i % 2 == 0 ? '$' : '%');
      }
    }
    // Aggiungo il terminatore di stringa
    righe[r][lun] = '\0';
    // Aggiorno il contatore delle righe lette
```

```
    r++;
  }

  // Chiude lo stream di input
  in.close();

  // Supponiamo che l'apertura del file di output
  // vada sempre a buon fine
  out.open(argv[3], ios::out);

  // Salva le righe al contrario
  for(int i = r - 1; i >= 0; i--) {
    out << righe[i] << endl;
  }

  // Chiude lo stream di output
  out.close();

  return 0;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti da riga di comando:

(a) il nome di un file di testo in input;
(b) un numero intero "**n**", corrispondente al numero di righe da leggere dal primo file;
(c) il nome di un file di testo in output.

legga le prime **n** righe del primo file, o tutte le righe del file se queste ultime sono meno di **n**, le processi come specificato sotto e poi salvi le righe risultanti **in ordine inverso** nel secondo file. Ogni riga deve essere riscritta come segue: ogni carattere separatore (' ', '\t') deve essere sostituito con il carattere '>' se si trova in una riga dispari, con il carattere '<' se si trova in una riga pari; ogni altro carattere deve rimanere invariato.

Utilizzare la funzione `atoi` della libreria `cstdlib` per leggere il numero di righe dal secondo parametro in ingresso.

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

```
La donzelletta vien dalla campagna,
in sul calar del sole,
col suo fascio dell'erba, e reca in mano
un mazzolin di rose e di viole.
(da "Il sabato del villaggio" di G. Leopardi)
```

allora il comando:

```
./a.out input.txt 4 output.txt
```

crea il file `output.txt` contenente il seguente testo:

```
un<mazzolin<di<rose<e<di<viole.
col>suo>fascio>dell'erba,>e>reca>in>mano
in<sul<calar<del<sole,
La>donzelletta>vien>dalla>campagna,
```

mentre il comando:

```
./a.out input.txt 6 output.txt
```

crea il file `output.txt` contenente il seguente testo:

```
(da>"Il>sabato>del>villaggio">di>G.>Leopardi)
un<mazzolin<di<rose<e<di<viole.
col>suo>fascio>dell'erba,>e>reca>in>mano
in<sul<calar<del<sole,
La>donzelletta>vien>dalla>campagna,
```

NOTA 1: Si assuma che nessuna riga del file di ingresso sia più lunga di **255** caratteri.

NOTA 2: Il programma non deve prevedere alcun numero massimo di parole o di righe leggibili dal file di input, pena l'annullamento dell'esercizio.

NOTA 3: È ammesso l'utilizzo delle funzioni contenute nelle librerie **cstring**.

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

1 **soluzione_A14.cc**

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

using namespace std;

const int DIMENSIONE_RIGA = 255 + 1;

int main(int argc, char * argv[]) {
  fstream in, out;
  int n_righe;
  char riga[DIMENSIONE_RIGA];
  char** righe;

  // Controllo argomenti passati in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_di_input> <numero_righe> <file_di_output>" << endl;
    exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  in.open(argv[1], ios::in);
  if (in.fail()) {
    cerr << "Il file " << argv[1] << " non esiste o non e' accessibile.\n";
    exit(EXIT_FAILURE);
  }

  // Legge il numero di righe dal parametro a riga di comando
  // (supponiamo sia sempre <= al numero effettivo di righe del file)
  n_righe = atoi(argv[2]);

  // Alloca lo spazio per le righe
  righe = new char* [n_righe];
  // Leggo il file di input, riga per riga
  int r = 0;
  while(in.getline(riga, DIMENSIONE_RIGA) && r < n_righe) {
    int lun = strlen(riga);
    // Alloca lo spazio per questa riga
    righe[r] = new char[lun + 1];
    for(int i = 0; i < lun; i++) {
      if(riga[i] != ' ' && riga[i] != '\t') {
        // Ricopio il carattere appena letto
        righe[r][i] = riga[i];
      } else {
        // Trasformo il carattere secondo l'algoritmo
        righe[r][i] = (r % 2 == 0 ? '>' : '<');
      }
    }
    // Aggiungo il terminatore di stringa
    righe[r][lun] = '\0';
    // Aggiorno il contatore delle righe lette
```

```
    r++;
  }

  // Chiude lo stream di input
  in.close();

  // Supponiamo che l'apertura del file di output
  // vada sempre a buon fine
  out.open(argv[3], ios::out);

  // Salva le righe al contrario
  for(int i = r - 1; i >= 0; i--) {
    out << righe[i] << endl;
  }

  // Chiude lo stream di output
  out.close();

  return 0;
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione **ricorsiva** `somma_prodotto_incrociato` che, presi come parametri due array di numeri interi `primo` e `secondo`, della stessa dimensione, e un terzo parametro intero `dim`, pari alla dimensione dei due array, restituisca la somma dei prodotti di elementi dei due array, calcolati come segue. I prodotti vanno calcolati moltiplicando il primo elemento del primo array con l'ultimo elemento del secondo, poi il secondo elemento del primo array con il penultimo del secondo, il terzo con il terzultimo e così via.

Per esempio, dati due array `a`:

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

e `b`:

```
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55}
```

così definiti il risultato del calcolo sara' **364**.

NOTA 1: La funzione deve essere **ricorsiva** ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Può fare uso di eventuali funzioni ausiliarie purchè **a loro volta ricorsive**.

NOTA 2: La funzione deve funzionare senza errore con ogni possibile array di dimensione uguale.

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

2 **codice_A21.cc**

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

// Inserire qui le DICHIARAZIONI delle funzioni

int main() {
  int primo[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int secondo[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

  cout << "La somma dei prodotti incrociati dei due array e' " <<
    somma_prodotto_incrociato(primo, secondo, 10) << endl;

  return 0;
}

// Inserire qui le DEFINIZIONI delle funzioni
```

2 **soluzione_A21.cc**

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

long somma_prodotto_incrociato(int a[], int b[], int dim);
long somma_prodotto_incrociato_ric(int a[], int b[], int primo, int ultimo);

int main() {
  int primo[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int secondo[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

  cout << "La somma dei prodotti incrociati dei due array e' " <<
    somma_prodotto_incrociato(primo, secondo, 10) << endl;

  return 0;
}

long somma_prodotto_incrociato(int a[], int b[], int dim) {
  return somma_prodotto_incrociato_ric(a, b, 0, dim - 1);
}

long somma_prodotto_incrociato_ric(int a[], int b[], int primo, int ultimo) {
  long ris = 0l;
  if(ultimo >= 0) {
    ris = (a[primo] * b[ultimo]) +
      somma_prodotto_incrociato_ric(a, b, primo + 1, ultimo - 1);
  }
  return ris;
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione **ricorsiva** `somma_prodotto_incrociato` che, presi come parametri due array di numeri interi `primo` e `secondo`, della stessa dimensione, e un terzo parametro intero `dim`, pari alla dimensione dei due array, restituisca la somma dei prodotti di elementi dei due array, calcolati come segue. I prodotti vanno calcolati moltiplicando il primo elemento del primo array con il secondo elemento del secondo, poi il secondo elemento del primo array con il terzo del secondo, il terzo elemento del primo con il quarto del secondo e così via; l'ultimo elemento del primo array va moltiplicato con il primo elemento del secondo.

Per esempio, dati due array `a`:

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

e `b`:

```
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55}
```

così definiti il risultato del calcolo sara' **1076**.

NOTA 1: La funzione deve essere **ricorsiva** ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Può fare uso di eventuali funzioni ausiliarie purchè **a loro volta ricorsive**.

NOTA 2: La funzione deve funzionare senza errore con ogni possibile array di dimensione uguale.

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

2 **codice_A22.cc**

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

// Inserire qui le DICHIARAZIONI delle funzioni

int main() {
  int primo[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int secondo[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

  cout << "La somma dei prodotti incrociati dei due array e' " <<
    somma_prodotto_incrociato(primo, secondo, 10) << endl;

  return 0;
}

// Inserire qui le DEFINIZIONI delle funzioni
```

2 **soluzione_A22.cc**

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

long somma_prodotto_incrociato(int a[], int b[], int dim);
long somma_prodotto_incrociato_ric(int a[], int b[], int indice, int dim);

int main() {
  int primo[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int secondo[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

  cout << "La somma dei prodotti incrociati dei due array e' " <<
    somma_prodotto_incrociato(primo, secondo, 10) << endl;

  return 0;
}

long somma_prodotto_incrociato(int a[], int b[], int dim) {
  return somma_prodotto_incrociato_ric(a, b, 0, dim);
}

long somma_prodotto_incrociato_ric(int a[], int b[], int indice, int dim) {
  long ris = 0l;
  if(indice < dim) {
    ris = (a[indice] * b[(indice + 1) % dim]) +
      somma_prodotto_incrociato_ric(a, b, indice + 1, dim);
  }
  return ris;
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione **ricorsiva** `prodotto_somma_incrociata` che, presi come parametri due array di numeri interi `primo` e `secondo`, della stessa dimensione, e un terzo parametro intero `dim`, pari alla dimensione dei due array, restituisca il prodotto della somma di elementi dei due array, calcolati come segue. Le somme vanno calcolate sommando il primo elemento del primo array con l'ultimo elemento del secondo, poi il secondo elemento del primo array con il penultimo del secondo, il terzo con il terzultimo e così via.

Per esempio, dati due array `a`:

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

e `b`:

```
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55}
```

così definiti il risultato del calcolo sara' **1293836544000**.

NOTA 1: La funzione deve essere **ricorsiva** ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Può fare uso di eventuali funzioni ausiliarie purchè **a loro volta ricorsive**.

NOTA 2: La funzione deve funzionare senza errore con ogni possibile array di dimensione uguale.

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

2 **codice_A23.cc**

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

// Inserire qui le DICHIARAZIONI delle funzioni

int main() {
  int primo[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int secondo[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

  cout << "Il prodotto delle somme incrociate dei due array e' " <<
    prodotto_somma_incrociata(primo, secondo, 10) << endl;

  return 0;
}

// Inserire qui le DEFINIZIONI delle funzioni
```

2 **soluzione_A23.cc**

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

long prodotto_somma_incrociata(int a[], int b[], int dim);
long prodotto_somma_incrociata_ric(int a[], int b[], int primo, int ultimo);

int main() {
  int primo[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int secondo[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

  cout << "Il prodotto delle somme incrociate dei due array e' " <<
    prodotto_somma_incrociata(primo, secondo, 10) << endl;

  return 0;
}

long prodotto_somma_incrociata(int a[], int b[], int dim) {
  return prodotto_somma_incrociata_ric(a, b, 0, dim - 1);
}

long prodotto_somma_incrociata_ric(int a[], int b[], int primo, int ultimo) {
  long ris = 1l;
  if(ultimo >= 0) {
    ris = (a[primo] + b[ultimo]) *
      prodotto_somma_incrociata_ric(a, b, primo + 1, ultimo - 1);
  }
  return ris;
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione **ricorsiva** `prodotto_somma_incrociata` che, presi come parametri due array di numeri interi `primo` e `secondo`, della stessa dimensione, e un terzo parametro intero `dim`, pari alla dimensione dei due array, restituisca il prodotto della somma di elementi dei due array, calcolati come segue. Le somme vanno calcolate sommando il primo elemento del primo array con il secondo elemento del secondo, poi il secondo elemento del primo array con il terzo del secondo, il terzo elemento del primo con il quarto del secondo e così via; l'ultimo elemento del primo array va sommato al primo elemento del secondo.

Per esempio, dati due array `a`:

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

e `b`:

```
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55}
```

così definiti il risultato del calcolo sara' **88340668416**.

NOTA 1: La funzione deve essere **ricorsiva** ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Può fare uso di eventuali funzioni ausiliarie purchè **a loro volta ricorsive**.

NOTA 2: La funzione deve funzionare senza errore con ogni possibile array di dimensione uguale.

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

2 **codice_A24.cc**

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

// Inserire qui le DICHIARAZIONI delle funzioni

int main() {
  int primo[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int secondo[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

  cout << "Il prodotto delle somme incrociate dei due array e' " <<
    prodotto_somma_incrociata(primo, secondo, 10) << endl;

  return 0;
}

// Inserire qui le DEFINIZIONI delle funzioni
```

2 **soluzione_A24.cc**

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

long prodotto_somma_incrociata(int a[], int b[], int dim);
long prodotto_somma_incrociata_ric(int a[], int b[], int indice, int dim);

int main() {
  int primo[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int secondo[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

  cout << "Il prodotto delle somme incrociate dei due array e' " <<
    prodotto_somma_incrociata(primo, secondo, 10) << endl;

  return 0;
}

long prodotto_somma_incrociata(int a[], int b[], int dim) {
  return prodotto_somma_incrociata_ric(a, b, 0, dim);
}

long prodotto_somma_incrociata_ric(int a[], int b[], int indice, int dim) {
  long ris = 1l;
  if(indice < dim) {
    ris = (a[indice] + b[(indice + 1) % dim]) *
      prodotto_somma_incrociata_ric(a, b, indice + 1, dim);
  }
  return ris;
}
```

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

- `init` inizializzi la coda;
- `deinit` liberi la memoria utilizzata dalla coda;
- `enqueue` inserisca l'elemento passato come parametro nella coda e restituisca `true` se l'operazione è andata a buon fine, `false` altrimenti;
- `dequeue` tolga l'elemento in testa alla coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `first` legga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `print` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente.

La coda deve essere implementata come una lista concatenata di puntatori allocata dinamicamente.

NOTA 1: tutte le operazioni implementate (ad eccezione di `deinit` e `print`) devono richiedere un numero costante di passi computazionali.

NOTA 2: non sono ammesse modifiche ai file `queue_main.cc` e `queue.h`.

NOTA 3: è ammessa la creazione di eventuali funzioni ausiliarie non dichiarate nel file header, purchè non vengano modificati i due file dati.

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

3 **queue_main.cc**

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

int main() {
  char res;
  float num;
  queue q;

  init(q);
  do {
    cout << "\nOperazioni possibili:\n"
         << "Enqueue (e)\n"
         << "Dequeue (d)\n"
         << "First (f)\n"
         << "Print (p)\n"
         << "Quit (q)\n";
    cin >> res;
    switch (res) {
    case 'e':
      cout << "Valore: ";
      cin >> num;
      if (!enqueue(q, num)) {
        cout << "Memoria piena!\n";
      }
      break;
    case 'd':
      if (!dequeue(q)) {
        cout << "Coda vuota\n";
      }
      break;
    case 'f':
      if (!first(q, num)) {
        cout << "Coda vuota!\n";
      } else {
        cout << "Primo elemento = " << num << endl;
      }
      break;
    case 'p':
      print(q);
      break;
    case 'q':
      break;
    default:
      cout << "Valore errato!\n";
    }
  } while (res != 'q');
  deinit(q);

  return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_QUEUE_H
#define STRUCT_QUEUE_H

struct elem {
  float value;
  elem* next;
};

struct queue {
  elem* head;
  elem* tail;
};

void init(queue &q);
void deinit(queue &q);
bool enqueue(queue &q, float n);
bool dequeue(queue &q);
bool first(const queue &q, float &out);
void print(const queue &q);

#endif
```

3 **queue_.cc**

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

void init(queue &q) {
  // Inizializza la coda
  q.tail = q.head = NULL;
}

void deinit(queue &q) {
  // Rimuove tutti gli elementi dalla coda
  while(dequeue(q));
  q.tail = NULL;
}

bool is_empty(const queue &q) {
  // Vero se e solo se la coda e' vuota
  return q.head == NULL;
}

bool enqueue(queue &q, float n) {
  bool ris = false;
  // Allocazione nuovo elemento
  elem* last = new (nothrow) elem;
  if (last != NULL) {
    last->next = NULL;
    last->value = n;
    if (is_empty(q)) {
```

```
      // Primo e ultimo elemento
      q.head = q.tail = last;
    } else {
      // Caso generale
      q.tail->next = last;
      q.tail = last;
    }
    ris = true;
  }
  return ris;
}

bool dequeue(queue &q) {
  bool ris = false;
  if (!is_empty(q)) {
    // Toglie il primo elemento dalla coda
    // e sposta il puntatore
    elem* first = q.head;
    q.head = q.head->next;
    delete first;
    ris = true;
  }
  return ris;
}

bool first(const queue &q, float &out) {
  bool ris = false;
  if (is_empty(q)) {
    out = q.head->value;
    ris = true;
  }
  return ris;
}

void print(const queue &q) {
  // Stampa tutti gli elementi
  elem* tmp = q.head;
  while (tmp != NULL) {
    cout << tmp->value << " ";
    tmp = tmp->next;
  }
  cout << endl;
}
```

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

- `init` inizializzi la coda;
- `deinit` liberi la memoria utilizzata dalla coda;
- `enqueue` inserisca l'elemento passato come parametro nella coda e restituisca `true` se l'operazione è andata a buon fine, `false` altrimenti;
- `dequeue` tolga l'elemento in testa alla coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `first` legga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `print` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente.

La coda deve essere implementata come una lista concatenata di puntatori allocata dinamicamente.

NOTA 1: tutte le operazioni implementate (ad eccezione di `deinit` e `print`) devono richiedere un numero costante di passi computazionali.

NOTA 2: non sono ammesse modifiche ai file `queue_main.cc` e `queue.h`.

NOTA 3: è ammessa la creazione di eventuali funzioni ausiliarie non dichiarate nel file header, purchè non vengano modificati i due file dati.

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

3 **queue_main.cc**

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

int main() {
  char res;
  int num;
  queue q;

  init(q);
  do {
    cout << "\nOperazioni possibili:\n"
         << "Enqueue (e)\n"
         << "Dequeue (d)\n"
         << "First (f)\n"
         << "Print (p)\n"
         << "Quit (q)\n";
    cin >> res;
    switch (res) {
    case 'e':
      cout << "Valore: ";
      cin >> num;
      if (!enqueue(q, num)) {
        cout << "Memoria piena!\n";
      }
      break;
    case 'd':
      if (!dequeue(q)) {
        cout << "Coda vuota\n";
      }
      break;
    case 'f':
      if (!first(q, num)) {
        cout << "Coda vuota!\n";
      } else {
        cout << "Primo elemento = " << num << endl;
      }
      break;
    case 'p':
      print(q);
      break;
    case 'q':
      break;
    default:
      cout << "Valore errato!\n";
    }
  } while (res != 'q');
  deinit(q);

  return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_QUEUE_H
#define STRUCT_QUEUE_H

struct elem {
  int value;
  elem* next;
};

struct queue {
  elem* head;
  elem* tail;
};

void init(queue &q);
void deinit(queue &q);
bool enqueue(queue &q, int n);
bool dequeue(queue &q);
bool first(const queue &q, int &out);
void print(const queue &q);

#endif
```

3 **queue_.cc**

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

void init(queue &q) {
  // Inizializza la coda
  q.tail = q.head = NULL;
}

void deinit(queue &q) {
  // Rimuove tutti gli elementi dalla coda
  while(dequeue(q));
  q.tail = NULL;
}

bool is_empty(const queue &q) {
  // Vero se e solo se la coda e' vuota
  return q.head == NULL;
}

bool enqueue(queue &q, int n) {
  bool ris = false;
  // Allocazione nuovo elemento
  elem* last = new (nothrow) elem;
  if (last != NULL) {
    last->next = NULL;
    last->value = n;
    if (is_empty(q)) {
```

```
      // Primo e ultimo elemento
      q.head = q.tail = last;
    } else {
      // Caso generale
      q.tail->next = last;
      q.tail = last;
    }
    ris = true;
  }
  return ris;
}

bool dequeue(queue &q) {
  bool ris = false;
  if (!is_empty(q)) {
    // Toglie il primo elemento dalla coda
    // e sposta il puntatore
    elem* first = q.head;
    q.head = q.head->next;
    delete first;
    ris = true;
  }
  return ris;
}

bool first(const queue &q, int &out) {
  bool ris = false;
  if (is_empty(q)) {
    out = q.head->value;
    ris = true;
  }
  return ris;
}

void print(const queue &q) {
  // Stampa tutti gli elementi
  elem* tmp = q.head;
  while (tmp != NULL) {
    cout << tmp->value << " ";
    tmp = tmp->next;
  }
  cout << endl;
}
```

3 Nel file `queue_main.cc` è definita la funzione `main` che contiene un menu per gestire una coda di `long`. Scrivere, in un nuovo file `queue.cc`, le definizioni delle funzioni dichiarate nello header file `queue.h` in modo tale che:

- `init` inizializzi la coda;
- `deinit` liberi la memoria utilizzata dalla coda;
- `enqueue` inserisca l'elemento passato come parametro nella coda e restituisca `true` se l'operazione è andata a buon fine, `false` altrimenti;
- `dequeue` tolga l'elemento in testa alla coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `first` legga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `print` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente.

La coda deve essere implementata come una lista concatenata di puntatori allocata dinamicamente.

NOTA 1: tutte le operazioni implementate (ad eccezione di `deinit` e `print`) devono richiedere un numero costante di passi computazionali.

NOTA 2: non sono ammesse modifiche ai file `queue_main.cc` e `queue.h`.

NOTA 3: è ammessa la creazione di eventuali funzioni ausiliarie non dichiarate nel file header, purchè non vengano modificati i due file dati.

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

3 **queue_main.cc**

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

int main() {
  char res;
  long num;
  queue q;

  init(q);
  do {
    cout << "\nOperazioni possibili:\n"
         << "Enqueue (e)\n"
         << "Dequeue (d)\n"
         << "First (f)\n"
         << "Print (p)\n"
         << "Quit (q)\n";
    cin >> res;
    switch (res) {
    case 'e':
      cout << "Valore: ";
      cin >> num;
      if (!enqueue(q, num)) {
        cout << "Memoria piena!\n";
      }
      break;
    case 'd':
      if (!dequeue(q)) {
        cout << "Coda vuota\n";
      }
      break;
    case 'f':
      if (!first(q, num)) {
        cout << "Coda vuota!\n";
      } else {
        cout << "Primo elemento = " << num << endl;
      }
      break;
    case 'p':
      print(q);
      break;
    case 'q':
      break;
    default:
      cout << "Valore errato!\n";
    }
  } while (res != 'q');
  deinit(q);

  return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_QUEUE_H
#define STRUCT_QUEUE_H

struct elem {
  long value;
  elem* next;
};

struct queue {
  elem* head;
  elem* tail;
};

void init(queue &q);
void deinit(queue &q);
bool enqueue(queue &q, long n);
bool dequeue(queue &q);
bool first(const queue &q, long &out);
void print(const queue &q);

#endif
```

3 **queue_.cc**

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

void init(queue &q) {
  // Inizializza la coda
  q.tail = q.head = NULL;
}

void deinit(queue &q) {
  // Rimuove tutti gli elementi dalla coda
  while(dequeue(q));
  q.tail = NULL;
}

bool is_empty(const queue &q) {
  // Vero se e solo se la coda e' vuota
  return q.head == NULL;
}

bool enqueue(queue &q, long n) {
  bool ris = false;
  // Allocazione nuovo elemento
  elem* last = new (nothrow) elem;
  if (last != NULL) {
    last->next = NULL;
    last->value = n;
    if (is_empty(q)) {
```

```
      // Primo e ultimo elemento
      q.head = q.tail = last;
    } else {
      // Caso generale
      q.tail->next = last;
      q.tail = last;
    }
    ris = true;
  }
  return ris;
}

bool dequeue(queue &q) {
  bool ris = false;
  if (!is_empty(q)) {
    // Toglie il primo elemento dalla coda
    // e sposta il puntatore
    elem* first = q.head;
    q.head = q.head->next;
    delete first;
    ris = true;
  }
  return ris;
}

bool first(const queue &q, long &out) {
  bool ris = false;
  if (is_empty(q)) {
    out = q.head->value;
    ris = true;
  }
  return ris;
}

void print(const queue &q) {
  // Stampa tutti gli elementi
  elem* tmp = q.head;
  while (tmp != NULL) {
    cout << tmp->value << " ";
    tmp = tmp->next;
  }
  cout << endl;
}
```

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

- `init` inizializzi la coda;
- `deinit` liberi la memoria utilizzata dalla coda;
- `enqueue` inserisca l'elemento passato come parametro nella coda e restituisca `true` se l'operazione è andata a buon fine, `false` altrimenti;
- `dequeue` tolga l'elemento in testa alla coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `first` legga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `print` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente.

La coda deve essere implementata come una lista concatenata di puntatori allocata dinamicamente.

NOTA 1: tutte le operazioni implementate (ad eccezione di `deinit` e `print`) devono richiedere un numero costante di passi computazionali.

NOTA 2: non sono ammesse modifiche ai file `queue_main.cc` e `queue.h`.

NOTA 3: è ammessa la creazione di eventuali funzioni ausiliarie non dichiarate nel file header, purchè non vengano modificati i due file dati.

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

3 **queue_main.cc**

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

int main() {
  char res;
  double num;
  queue q;

  init(q);
  do {
    cout << "\nOperazioni possibili:\n"
         << "Enqueue (e)\n"
         << "Dequeue (d)\n"
         << "First (f)\n"
         << "Print (p)\n"
         << "Quit (q)\n";
    cin >> res;
    switch (res) {
    case 'e':
      cout << "Valore: ";
      cin >> num;
      if (!enqueue(q, num)) {
        cout << "Memoria piena!\n";
      }
      break;
    case 'd':
      if (!dequeue(q)) {
        cout << "Coda vuota\n";
      }
      break;
    case 'f':
      if (!first(q, num)) {
        cout << "Coda vuota!\n";
      } else {
        cout << "Primo elemento = " << num << endl;
      }
      break;
    case 'p':
      print(q);
      break;
    case 'q':
      break;
    default:
      cout << "Valore errato!\n";
    }
  } while (res != 'q');
  deinit(q);

  return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_QUEUE_H
#define STRUCT_QUEUE_H

struct elem {
  double value;
  elem* next;
};

struct queue {
  elem* head;
  elem* tail;
};

void init(queue &q);
void deinit(queue &q);
bool enqueue(queue &q, double n);
bool dequeue(queue &q);
bool first(const queue &q, double &out);
void print(const queue &q);

#endif
```

3 **queue_.cc**

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

void init(queue &q) {
  // Inizializza la coda
  q.tail = q.head = NULL;
}

void deinit(queue &q) {
  // Rimuove tutti gli elementi dalla coda
  while(dequeue(q));
  q.tail = NULL;
}

bool is_empty(const queue &q) {
  // Vero se e solo se la coda e' vuota
  return q.head == NULL;
}

bool enqueue(queue &q, double n) {
  bool ris = false;
  // Allocazione nuovo elemento
  elem* last = new (nothrow) elem;
  if (last != NULL) {
    last->next = NULL;
    last->value = n;
    if (is_empty(q)) {
```

```
      // Primo e ultimo elemento
      q.head = q.tail = last;
    } else {
      // Caso generale
      q.tail->next = last;
      q.tail = last;
    }
    ris = true;
  }
  return ris;
}

bool dequeue(queue &q) {
  bool ris = false;
  if (!is_empty(q)) {
    // Toglie il primo elemento dalla coda
    // e sposta il puntatore
    elem* first = q.head;
    q.head = q.head->next;
    delete first;
    ris = true;
  }
  return ris;
}

bool first(const queue &q, double &out) {
  bool ris = false;
  if (is_empty(q)) {
    out = q.head->value;
    ris = true;
  }
  return ris;
}

void print(const queue &q) {
  // Stampa tutti gli elementi
  elem* tmp = q.head;
  while (tmp != NULL) {
    cout << tmp->value << " ";
    tmp = tmp->next;
  }
  cout << endl;
}
```

4 Sono dati:

(a) le definizioni di albero di ricerca binaria e relativi header nel file **tree.h**;

(b) le definizioni delle primitive di cui sopra, ad eccezione della primitiva `stampa`, nel file oggetto **tree.o**;

(c) una funzione `main()` che le utilizza, nel file **tree_main.cc**.

Scrivere in fondo al file **tree_main.cc** una realizzazione **NON RICORSIVA** (e non **MUTUALMENTE RICORSIVA**) della funzione `stampa`.

La realizzazione deve poter funzionare per qualsiasi dimensione e tipo di albero immesso dall'utente.

E' possibile utilizzare strutture dati ausiliarie viste a lezione e relative primitive, purche' a loro volta non ricorsive o mutualmente ricorsive.

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

4 **soluzione_A41.cc**

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

using namespace std;

void stampa(const tree &);

int main()
{
  char option,val;
  tree t, tmp;
  retval res;
  init(t);
  do {
    cout << "\nOperazioni possibili:\n"
         << "Inserimento (i)\n"
         << "Ricerca (r)\n"
         << "Stampa ordinata (s)\n"
         << "Fine (f)\n";
    cin >> option;
    switch (option) {
    case 'i':
      cout << "Val? : ";
      cin >> val;
      res = insert(t,val);
      if (res == FAIL)
        cout << "spazio insufficiente!\n";
      break;
    case 'r':
      cout << "Val? : ";
      cin >> val;
      tmp=cerca(t,val);
      if (!nullp(tmp))
        cout << "Valore trovato!: " << val << endl;
      else
        cout << "Valore non trovato!\n";
      break;
    case 's':
      cout << "Stampa ordinata:\n";
      stampa(t);
      break;
    case 'f':
      break;
    default:
      cout << "Opzione errata\n";
    }
  } while (option != 'f');
}

// Si scrivano qui sotto le definizioni della funzione *NON RICORSIVA*:
// void stampa(const tree &)
// e di tutte le funzioni e strutture dati ausiliarie richieste.
```

```
// Definizione di uno stack ausiliario di tree

struct stacknode{
  tree val;
  stacknode *next;
};

typedef stacknode * stack ;


bool emptyp (const stack & s) {
  return (s == NULL);
}

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

retval top (tree &n,const stack & s) {
  retval res;
  if (emptyp(s))
    res=FAIL;
  else {
    n=s -> val;
    res=OK;
  }
  return res;
}

retval push (tree n,stack & s) {
  retval res;
  stacknode * np = new (nothrow) stacknode;
  if (np==NULL)
    res = FAIL;
  else {
    np -> val = n;
    np -> next = s;
    s = np;
    res = OK;
  }
  return res;
}

retval pop (stack & s) {
  retval res;
  if (emptyp(s))
    res=FAIL;
  else {
    stacknode *first = s;
    s = s -> next;
    delete first;
    res=OK;
  }
```

```
  return res;
}


// DEFINIZIONE DELLA ROUTINE ITERATIVA DI STAMPA

void stampa(const tree & t) {
  tree tmp = t;
  stack s;
  init(s);
  while (!emptyp(s) || tmp!=NULL) {
    if (tmp!=NULL) {
      push(tmp,s);
      tmp = tmp->left;
    }
    else {
      top(tmp,s);
      pop(s);
      cout << tmp->item << endl;
      tmp=tmp->right;
    }
  }
}
```