Primo Appello di Programmazione I

13 Gennaio 2020
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 Dati **due** file di testo contenenti elenchi ordinati di parole, scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi dei due file in input e il nome di un terzo file di output, salvi in quest'ultimo solo le parole presenti sia nel primo che nel secondo file. Si assuma per semplicità che nessuna parola sia più lunga di `255 caratteri`.

Si noti che:

- il primo e il secondo file di input contengono un **elenco ordinato in senso crescente** di parole, una per riga. Ogni riga dei file contiene quindi un solo termine, ad eccezione della prima riga che contiene il numero delle parole del file stesso; questa prima parola della prima riga va comunque esclusa dal conteggio. **Non è possibile fare a priori alcuna assunzione sul numero di righe di questi file (possono contenere un numero di righe grande a piacere.)**
- la sintassi del terzo file è identica a quella dei precedenti, salvo la mancanza dell'indicazione del numero di parole del file nella prima riga.

Ecco un esempio del contenuto rispettivamente del primo, del secondo e del terzo file:

10 abaco barca categoria elicottero matematica navigare opera quattordici sotto stella

7 carta elicottero navigare opera oro quattordici sopra

elicottero navigare opera quattordici

NOTA 1: Il programma deve terminare con un messaggio appropriato qualora il numero di argomenti sulla linea di comando non sia corretto, o qualora uno dei file di input risulti non accessibile.

NOTA 2: Non è ammesso aprire più di una volta ciascun file durante l'esecuzione del programma: la lettura di ciascuno stream deve essere **sequenziale**, dall'inizio alla fine.

NOTA 3: Viene sottratto un punto (-1) alla valutazione dell'esercizio se la complessità dell'algoritmo è superiore, nel caso peggiore, a $O(m \cdot log(n))$ oppure $O(n \cdot log(m))$, dove n, m sono il numero di parole contenute nel primo e del secondo file rispettivamente.

NOTA 4: È possibile impiegare le funzioni della libreria `cstring` e `iostream` viste a lezione.

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

1 **esercizio1.cc**

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

using namespace std;

const int MAX_CHAR = 255 + 1;

// Funzioni di utilita'
bool appartiene(char* s, char* lista[], int quanti);
bool appartiene(char* s, char* lista[], int inf, int sup);

int main(int argc,char* argv[]){
  fstream file_a, file_b, file_output;
  int maxnum;
  char parola[MAX_CHAR];

  // Controllo parametri in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_a> <file_b> <risultato>\n";
    exit(-1);
  }

  // Apro il primo file
  file_a.open(argv[1],ios::in);
  // Controllo apertura primo file
  if(file_a.fail()) {
    cerr << "Errore apertura primo file di input\n";
    exit(-1);
  }
  // Numero righe del file
  file_a >> maxnum;
  // Allocazione dinamica
  char** parole = new char* [maxnum];
  int i;
  // Prima parola del file
  file_a >> parola;
  for(i = 0; !file_a.eof(); i++) {
    // Allocazione dinamica
    parole[i] = new char[strlen(parola) + 1];
    // Non e' necessario utilizzare strncpy,
    // perche' "parola" e', per costruzione,
    // una stringa "ben formata"
    strcpy(parole[i], parola);
    // Continua lettura file
    file_a >> parola;
  }
  // A questo punto, file_a non serve piu'
  file_a.close();

  // Apro il secondo file
```

```
  file_b.open(argv[2],ios::in);
  // Controllo apertura secondo file
  if(file_b.fail()) {
    cerr << "Errore apertura secondo file di input\n";
    exit(-1);
  }
  // Apro file di output
  file_output.open(argv[3],ios::out);
  // Leggo e scarto il numero righe del secondo file
  // perche' non lo utilizzo
  file_b >> i;
  // Prima parola del file
  file_b >> parola;
  while(!file_b.eof()) {
    // Se parola e' presente anche in file_a,
    // allora la salva nel file di output
    if(appartiene(parola, parole, maxnum)) {
      file_output << parola << endl;
    }
    // Continua lettura file
    file_b >> parola;
  }
  // Chiusura stream secondo file
  file_b.close();
  // Chiusura stream file di output
  file_output.close();

  return(0);
}

bool appartiene(char* s, char* lista[], int quanti) {
  bool trovato = false;
  // Ricerca lineare (funziona, ma vale
  // un punto in meno)
  /*
  int i = 0;
  while(!trovato && i < quanti) {
    if(strcmp(s, lista[i]) == 0) {
      trovato = true;
    }
    i++;
  }
  */
  // Ricerca per bisezione (ricorsiva)
  trovato = appartiene(s, lista, 0, quanti - 1);
  // Ritorno il risultato
  return trovato;
}

bool appartiene(char* s, char* lista[], int inf, int sup) {
  bool ris = false;
  if(inf <= sup) {
    int mid = (inf + sup) / 2;
    int cmp = strcmp(s, lista[mid]);
```

```
    if(cmp == 0) {
      ris = true;
    } else if(cmp < 0) {
      ris = appartiene(s, lista, inf, mid - 1);
    } else {
      ris = appartiene(s, lista, mid + 1, sup);
    }
  }
  return ris;
}
```

1 **esercizio1.cc**

```
// Soluzione senza allocazione dinamica

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

using namespace std;

const int MAX_CHAR = 255 + 1;

int main(int argc,char* argv[]){
  fstream file_a, file_b, file_output;
  int maxnum_a, maxnum_b;
  char parola_a[MAX_CHAR], parola_b[MAX_CHAR];

  // Controllo parametri in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_a> <file_b> <risultato>\n";
    exit(-1);
  }

  // Apro il primo file
  file_a.open(argv[1],ios::in);
  // Controllo apertura primo file
  if(file_a.fail()) {
    cerr << "Errore apertura primo file di input\n";
    exit(-1);
  }
  // Numero righe del primo file
  file_a >> maxnum_a;
  // Apro il secondo file
  file_b.open(argv[2],ios::in);
  // Controllo apertura secondo file
  if(file_b.fail()) {
    cerr << "Errore apertura secondo file di input\n";
    exit(-1);
  }
  // Numero righe del secondo file
  file_b >> maxnum_b;
  // Apro file di output
  file_output.open(argv[3],ios::out);

  // Prima parola del primo file
  file_a >> parola_a;
  // Prima parola del secondo file
  file_b >> parola_b;
  for(int indice_a = 0, indice_b = 0;
      indice_a < maxnum_a && indice_b < maxnum_b;) {
    // Confronto tra le parole lette nei due file
    int test = strcmp(parola_a, parola_b);
    // Verifica risultato test
```

```
    if(test == 0) {
      // Sono uguali; allora la parola va salvata
      file_output << parola_a << endl;
      // E vanno letti entrambi i file
      file_a >> parola_a;
      file_b >> parola_b;
      // E aggiornati entrambi gli indici
      indice_a++;
      indice_b++;
    } else if(test < 0) {
      // "parola_a" precede "parola_b"; allora
      // leggo dal primo file
      file_a >> parola_a;
      // E aggiorno l'indice
      indice_a++;
    } else {
      // "parola_b" precede "parola_a"; allora
      // leggo dal secondo file
      file_b >> parola_b;
      // E aggiorno l'indice
      indice_b++;
    }
  }

  // Chiusura stream primo file
  file_a.close();
  // Chiusura stream secondo file
  file_b.close();
  // Chiusura stream file di output
  file_output.close();

  return(0);
}
```

1 Dati **due** file di testo contenenti elenchi ordinati di parole, scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi dei due file in input e il nome di un terzo file di output, salvi in quest'ultimo solo le parole presenti solo nel primo ma non nel secondo file. Si assuma per semplicità che nessuna parola sia più lunga di `255 caratteri`.

Si noti che:

- il primo e il secondo file di input contengono un **elenco ordinato in senso crescente** di parole, una per riga. Ogni riga dei file contiene quindi un solo termine, ad eccezione della prima riga che contiene il numero delle parole del file stesso; questa prima parola della prima riga va comunque esclusa dal conteggio. **Non è possibile fare a priori alcuna assunzione sul numero di righe di questi file (possono contenere un numero di righe grande a piacere.)**
- la sintassi del terzo file è identica a quella dei precedenti, salvo la mancanza dell'indicazione del numero di parole del file nella prima riga.

Ecco un esempio del contenuto rispettivamente del primo, del secondo e del terzo file:

10 abaco barca categoria elicottero matematica navigare opera quattordici sotto stella

7 carta elicottero navigare opera oro quattordici sopra

abaco barca categoria matematica sotto stella

NOTA 1: Il programma deve terminare con un messaggio appropriato qualora il numero di argomenti sulla linea di comando non sia corretto, o qualora uno dei file di input risulti non accessibile.

NOTA 2: Non è ammesso aprire più di una volta ciascun file durante l'esecuzione del programma: la lettura di ciascuno stream deve essere **sequenziale**, dall'inizio alla fine.

NOTA 3: Viene sottratto un punto (-1) alla valutazione dell'esercizio se la complessità dell'algoritmo è superiore, nel caso peggiore, a $O(m \cdot log(n))$ oppure $O(n \cdot log(m))$, dove n, m sono il numero di parole contenute nel primo e del secondo file rispettivamente.

NOTA 4: È possibile impiegare le funzioni della libreria `cstring` e `iostream` viste a lezione.

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

1 **esercizio1.cc**

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

using namespace std;

const int MAX_CHAR = 255 + 1;

// Funzioni di utilita'
bool appartiene(char* s, char* lista[], int quanti);
bool appartiene(char* s, char* lista[], int inf, int sup);

int main(int argc,char* argv[]){
  fstream file_a, file_b, file_output;
  int maxnum;
  char parola[MAX_CHAR];

  // Controllo parametri in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_a> <file_b> <risultato>\n";
    exit(-1);
  }

  // Apro il secondo file
  file_b.open(argv[2],ios::in);
  // Controllo apertura secondo file
  if(file_b.fail()) {
    cerr << "Errore apertura secondo file di input\n";
    exit(-1);
  }
  // Numero righe del file
  file_b >> maxnum;
  // Allocazione dinamica
  char** parole = new char* [maxnum];
  int i;
  // Prima parola del file
  file_b >> parola;
  for(i = 0; !file_b.eof(); i++) {
    // Allocazione dinamica
    parole[i] = new char[strlen(parola) + 1];
    // Non e' necessario utilizzare strncpy,
    // perche' "parola" e', per costruzione,
    // una stringa "ben formata"
    strcpy(parole[i], parola);
    // Continua lettura file
    file_b >> parola;
  }
  // A questo punto, file_b non serve piu'
  file_b.close();

  // Apro il primo file
```

```
  file_a.open(argv[1],ios::in);
  // Controllo apertura primo file
  if(file_a.fail()) {
    cerr << "Errore apertura primo file di input\n";
    exit(-1);
  }
  // Apro file di output
  file_output.open(argv[3],ios::out);
  // Leggo e scarto il numero righe del secondo file
  // perche' non lo utilizzo
  file_a >> i;
  // Prima parola del file
  file_a >> parola;
  while(!file_a.eof()) {
    // Se parola e' presente anche in file_b,
    // allora non la salva nel file di output
    if(!appartiene(parola, parole, maxnum)) {
      file_output << parola << endl;
    }
    // Continua lettura file
    file_a >> parola;
  }
  // Chiusura stream secondo file
  file_a.close();
  // Chiusura stream file di output
  file_output.close();

  return(0);
}

bool appartiene(char* s, char* lista[], int quanti) {
  bool trovato = false;
  // Ricerca lineare (funziona, ma vale
  // un punto in meno)
  /*
  int i = 0;
  while(!trovato && i < quanti) {
    if(strcmp(s, lista[i]) == 0) {
      trovato = true;
    }
    i++;
  }
  */
  // Ricerca per bisezione (ricorsiva)
  trovato = appartiene(s, lista, 0, quanti - 1);
  // Ritorno il risultato
  return trovato;
}

bool appartiene(char* s, char* lista[], int inf, int sup) {
  bool ris = false;
  if(inf <= sup) {
    int mid = (inf + sup) / 2;
    int cmp = strcmp(s, lista[mid]);
```

```
    if(cmp == 0) {
      ris = true;
    } else if(cmp < 0) {
      ris = appartiene(s, lista, inf, mid - 1);
    } else {
      ris = appartiene(s, lista, mid + 1, sup);
    }
  }
  return ris;
}
```

1 **esercizio1.cc**

```
// Soluzione senza allocazione dinamica

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

using namespace std;

const int MAX_CHAR = 255 + 1;

int main(int argc,char* argv[]){
  fstream file_a, file_b, file_output;
  int maxnum_a, maxnum_b;
  char parola_a[MAX_CHAR], parola_b[MAX_CHAR];

  // Controllo parametri in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_a> <file_b> <risultato>\n";
    exit(-1);
  }

  // Apro il primo file
  file_a.open(argv[1],ios::in);
  // Controllo apertura primo file
  if(file_a.fail()) {
    cerr << "Errore apertura primo file di input\n";
    exit(-1);
  }
  // Numero righe del primo file
  file_a >> maxnum_a;
  // Apro il secondo file
  file_b.open(argv[2],ios::in);
  // Controllo apertura secondo file
  if(file_b.fail()) {
    cerr << "Errore apertura secondo file di input\n";
    exit(-1);
  }
  // Numero righe del secondo file
  file_b >> maxnum_b;
  // Apro file di output
  file_output.open(argv[3],ios::out);

  // Prima parola del primo file
  file_a >> parola_a;
  // Prima parola del secondo file
  file_b >> parola_b;
  for(int indice_a = 0, indice_b = 0;
      indice_a < maxnum_a;) {
    // Confronto tra le parole lette nei due file
    int test = strcmp(parola_a, parola_b);
    // Verifica risultato test
```

```
    if(test == 0) {
      // Sono uguali; allora la parola non va salvata,
      // vanno solo letti entrambi i file
      file_a >> parola_a;
      file_b >> parola_b;
      // E aggiornati entrambi gli indici
      indice_a++;
      indice_b++;
    } else if(test < 0 || indice_b >= maxnum_b) {
      // "parola_a" precede "parola_b", oppure
      // "file_b" e' terminato; allora
      // salvo la parola nel file di output
      file_output << parola_a << endl;
      // E leggo dal primo file
      file_a >> parola_a;
      // E aggiorno l'indice
      indice_a++;
    } else {
      // "parola_b" precede "parola_a"; allora
      // leggo dal secondo file
      file_b >> parola_b;
      // E aggiorno l'indice
      indice_b++;
    }
  }

  // Chiusura stream primo file
  file_a.close();
  // Chiusura stream secondo file
  file_b.close();
  // Chiusura stream file di output
  file_output.close();

  return(0);
}
```

1 Dati **due** file di testo contenenti elenchi ordinati di parole, scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi dei due file in input e il nome di un terzo file di output, salvi in quest'ultimo solo le parole presenti sia nel primo che nel secondo file. Si assuma per semplicità che nessuna parola sia più lunga di `255 caratteri`.

Si noti che:

- il primo e il secondo file di input contengono un **elenco ordinato in senso decrescente** di parole, una per riga. Ogni riga dei file contiene quindi un solo termine, ad eccezione della prima riga che contiene il numero delle parole del file stesso; questa prima parola della prima riga va comunque esclusa dal conteggio. **Non è possibile fare a priori alcuna assunzione sul numero di righe di questi file (possono contenere un numero di righe grande a piacere.)**
- la sintassi del terzo file è identica a quella dei precedenti, salvo la mancanza dell'indicazione del numero di parole del file nella prima riga.

Ecco un esempio del contenuto rispettivamente del primo, del secondo e del terzo file:

10
zappa
terrazzo
radice
quanti
quadro
mirtillo
lista
falena
domino
apice

7
utile
terrazzo
mirtillo
giostra
dono
domino
apice

terrazzo
mirtillo
domino
apice

NOTA 1: Il programma deve terminare con un messaggio appropriato qualora il numero di argomenti sulla linea di comando non sia corretto, o qualora uno dei file di input risulti non accessibile.

NOTA 2: Non è ammesso aprire più di una volta ciascun file durante l'esecuzione del programma: la lettura di ciascuno stream deve essere **sequenziale**, dall'inizio alla fine.

NOTA 3: Viene sottratto un punto (-1) alla valutazione dell'esercizio se la complessità dell'algoritmo è superiore, nel caso peggiore, a $O(m \cdot log(n))$ oppure $O(n \cdot log(m))$, dove n, m sono il numero di parole contenute nel primo e del secondo file rispettivamente.

NOTA 4: È possibile impiegare le funzioni della libreria `cstring` e `iostream` viste a lezione.

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

1 **esercizio1.cc**

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

using namespace std;

const int MAX_CHAR = 255 + 1;

// Funzioni di utilita'
bool appartiene(char* s, char* lista[], int quanti);
bool appartiene(char* s, char* lista[], int inf, int sup);

int main(int argc,char* argv[]){
  fstream file_a, file_b, file_output;
  int maxnum;
  char parola[MAX_CHAR];

  // Controllo parametri in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_a> <file_b> <risultato>\n";
    exit(-1);
  }

  // Apro il primo file
  file_a.open(argv[1],ios::in);
  // Controllo apertura primo file
  if(file_a.fail()) {
    cerr << "Errore apertura primo file di input\n";
    exit(-1);
  }
  // Numero righe del file
  file_a >> maxnum;
  // Allocazione dinamica
  char** parole = new char* [maxnum];
  int i;
  // Prima parola del file
  file_a >> parola;
  for(i = 0; !file_a.eof(); i++) {
    // Allocazione dinamica
    parole[i] = new char[strlen(parola) + 1];
    // Non e' necessario utilizzare strncpy,
    // perche' "parola" e', per costruzione,
    // una stringa "ben formata"
    strcpy(parole[i], parola);
    // Continua lettura file
    file_a >> parola;
  }
  // A questo punto, file_a non serve piu'
  file_a.close();

  // Apro il secondo file
```

```
  file_b.open(argv[2],ios::in);
  // Controllo apertura secondo file
  if(file_b.fail()) {
    cerr << "Errore apertura secondo file di input\n";
    exit(-1);
  }
  // Apro file di output
  file_output.open(argv[3],ios::out);
  // Leggo e scarto il numero righe del secondo file
  // perche' non lo utilizzo
  file_b >> i;
  // Prima parola del file
  file_b >> parola;
  while(!file_b.eof()) {
    // Se parola e' presente anche in file_a,
    // allora la salva nel file di output
    if(appartiene(parola, parole, maxnum)) {
      file_output << parola << endl;
    }
    // Continua lettura file
    file_b >> parola;
  }
  // Chiusura stream secondo file
  file_b.close();
  // Chiusura stream file di output
  file_output.close();

  return(0);
}

bool appartiene(char* s, char* lista[], int quanti) {
  bool trovato = false;
  // Ricerca lineare (funziona, ma vale
  // un punto in meno)
  /*
  int i = 0;
  while(!trovato && i < quanti) {
    if(strcmp(s, lista[i]) == 0) {
      trovato = true;
    }
    i++;
  }
  */
  // Ricerca per bisezione (ricorsiva)
  trovato = appartiene(s, lista, 0, quanti - 1);
  // Ritorno il risultato
  return trovato;
}

bool appartiene(char* s, char* lista[], int inf, int sup) {
  bool ris = false;
  if(inf <= sup) {
    int mid = (inf + sup) / 2;
    int cmp = strcmp(s, lista[mid]);
```

```
    if(cmp == 0) {
      ris = true;
    } else if(cmp > 0) {
      ris = appartiene(s, lista, inf, mid - 1);
    } else {
      ris = appartiene(s, lista, mid + 1, sup);
    }
  }
  return ris;
}
```

1 **esercizio1.cc**

```
// Soluzione senza allocazione dinamica

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

using namespace std;

const int MAX_CHAR = 255 + 1;

int main(int argc,char* argv[]){
  fstream file_a, file_b, file_output;
  int maxnum_a, maxnum_b;
  char parola_a[MAX_CHAR], parola_b[MAX_CHAR];

  // Controllo parametri in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_a> <file_b> <risultato>\n";
    exit(-1);
  }

  // Apro il primo file
  file_a.open(argv[1],ios::in);
  // Controllo apertura primo file
  if(file_a.fail()) {
    cerr << "Errore apertura primo file di input\n";
    exit(-1);
  }
  // Numero righe del primo file
  file_a >> maxnum_a;
  // Apro il secondo file
  file_b.open(argv[2],ios::in);
  // Controllo apertura secondo file
  if(file_b.fail()) {
    cerr << "Errore apertura secondo file di input\n";
    exit(-1);
  }
  // Numero righe del secondo file
  file_b >> maxnum_b;
  // Apro file di output
  file_output.open(argv[3],ios::out);

  // Prima parola del primo file
  file_a >> parola_a;
  // Prima parola del secondo file
  file_b >> parola_b;
  for(int indice_a = 0, indice_b = 0;
      indice_a < maxnum_a && indice_b < maxnum_b;) {
    // Confronto tra le parole lette nei due file
    int test = strcmp(parola_a, parola_b);
    // Verifica risultato test
```

```
    if(test == 0) {
      // Sono uguali; allora la parola va salvata
      file_output << parola_a << endl;
      // E vanno letti entrambi i file
      file_a >> parola_a;
      file_b >> parola_b;
      // E aggiornati entrambi gli indici
      indice_a++;
      indice_b++;
    } else if(test > 0) {
      // "parola_a" segue "parola_b"; allora
      // leggo dal primo file
      file_a >> parola_a;
      // E aggiorno l'indice
      indice_a++;
    } else {
      // "parola_b" segue "parola_a"; allora
      // leggo dal secondo file
      file_b >> parola_b;
      // E aggiorno l'indice
      indice_b++;
    }
  }

  // Chiusura stream primo file
  file_a.close();
  // Chiusura stream secondo file
  file_b.close();
  // Chiusura stream file di output
  file_output.close();

  return(0);
}
```

1 Dati **due** file di testo contenenti elenchi ordinati di parole, scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi dei due file in input e il nome di un terzo file di output, salvi in quest'ultimo solo le parole presenti solo nel primo ma non nel secondo file. Si assuma per semplicità che nessuna parola sia più lunga di `255 caratteri`.

Si noti che:

- il primo e il secondo file di input contengono un **elenco ordinato in senso decrescente** di parole, una per riga. Ogni riga dei file contiene quindi un solo termine, ad eccezione della prima riga che contiene il numero delle parole del file stesso; questa prima parola della prima riga va comunque esclusa dal conteggio. **Non è possibile fare a priori alcuna assunzione sul numero di righe di questi file (possono contenere un numero di righe grande a piacere.)**
- la sintassi del terzo file è identica a quella dei precedenti, salvo la mancanza dell'indicazione del numero di parole del file nella prima riga.

Ecco un esempio del contenuto rispettivamente del primo, del secondo e del terzo file:

10
zappa
terrazzo
radice
quanti
quadro
mirtillo
lista
falena
domino
apice

7
utile
terrazzo
mirtillo
giostra
dono
domino
apice

zappa
radice
quanti
quadro
lista
falena

NOTA 1: Il programma deve terminare con un messaggio appropriato qualora il numero di argomenti sulla linea di comando non sia corretto, o qualora uno dei file di input risulti non accessibile.

NOTA 2: Non è ammesso aprire più di una volta ciascun file durante l'esecuzione del programma: la lettura di ciascuno stream deve essere **sequenziale**, dall'inizio alla fine.

NOTA 3: Viene sottratto un punto (-1) alla valutazione dell'esercizio se la complessità dell'algoritmo è superiore, nel caso peggiore, a $O(m \cdot log(n))$ oppure $O(n \cdot log(m))$, dove n, m sono il numero di parole contenute nel primo e del secondo file rispettivamente.

NOTA 4: È possibile impiegare le funzioni della libreria `cstring` e `iostream` viste a lezione.

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

1 **esercizio1.cc**

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

using namespace std;

const int MAX_CHAR = 255 + 1;

// Funzioni di utilita'
bool appartiene(char* s, char* lista[], int quanti);
bool appartiene(char* s, char* lista[], int inf, int sup);

int main(int argc,char* argv[]){
  fstream file_a, file_b, file_output;
  int maxnum;
  char parola[MAX_CHAR];

  // Controllo parametri in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_a> <file_b> <risultato>\n";
    exit(-1);
  }

  // Apro il secondo file
  file_b.open(argv[2],ios::in);
  // Controllo apertura secondo file
  if(file_b.fail()) {
    cerr << "Errore apertura secondo file di input\n";
    exit(-1);
  }
  // Numero righe del file
  file_b >> maxnum;
  // Allocazione dinamica
  char** parole = new char* [maxnum];
  int i;
  // Prima parola del file
  file_b >> parola;
  for(i = 0; !file_b.eof(); i++) {
    // Allocazione dinamica
    parole[i] = new char[strlen(parola) + 1];
    // Non e' necessario utilizzare strncpy,
    // perche' "parola" e', per costruzione,
    // una stringa "ben formata"
    strcpy(parole[i], parola);
    // Continua lettura file
    file_b >> parola;
  }
  // A questo punto, file_b non serve piu'
  file_b.close();

  // Apro il primo file
```

```
  file_a.open(argv[1],ios::in);
  // Controllo apertura primo file
  if(file_a.fail()) {
    cerr << "Errore apertura primo file di input\n";
    exit(-1);
  }
  // Apro file di output
  file_output.open(argv[3],ios::out);
  // Leggo e scarto il numero righe del secondo file
  // perche' non lo utilizzo
  file_a >> i;
  // Prima parola del file
  file_a >> parola;
  while(!file_a.eof()) {
    // Se parola e' presente anche in file_b,
    // allora non la salva nel file di output
    if(!appartiene(parola, parole, maxnum)) {
      file_output << parola << endl;
    }
    // Continua lettura file
    file_a >> parola;
  }
  // Chiusura stream secondo file
  file_a.close();
  // Chiusura stream file di output
  file_output.close();

  return(0);
}

bool appartiene(char* s, char* lista[], int quanti) {
  bool trovato = false;
  // Ricerca lineare (funziona, ma vale
  // un punto in meno)
  /*
  int i = 0;
  while(!trovato && i < quanti) {
    if(strcmp(s, lista[i]) == 0) {
      trovato = true;
    }
    i++;
  }
  */
  // Ricerca per bisezione (ricorsiva)
  trovato = appartiene(s, lista, 0, quanti - 1);
  // Ritorno il risultato
  return trovato;
}

bool appartiene(char* s, char* lista[], int inf, int sup) {
  bool ris = false;
  if(inf <= sup) {
    int mid = (inf + sup) / 2;
    int cmp = strcmp(s, lista[mid]);
```

```
    if(cmp == 0) {
      ris = true;
    } else if(cmp > 0) {
      ris = appartiene(s, lista, inf, mid - 1);
    } else {
      ris = appartiene(s, lista, mid + 1, sup);
    }
  }
  return ris;
}
```

1 **esercizio1.cc**

```
// Soluzione senza allocazione dinamica

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

using namespace std;

const int MAX_CHAR = 255 + 1;

int main(int argc,char* argv[]){
  fstream file_a, file_b, file_output;
  int maxnum_a, maxnum_b;
  char parola_a[MAX_CHAR], parola_b[MAX_CHAR];

  // Controllo parametri in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_a> <file_b> <risultato>\n";
    exit(-1);
  }

  // Apro il primo file
  file_a.open(argv[1],ios::in);
  // Controllo apertura primo file
  if(file_a.fail()) {
    cerr << "Errore apertura primo file di input\n";
    exit(-1);
  }
  // Numero righe del primo file
  file_a >> maxnum_a;
  // Apro il secondo file
  file_b.open(argv[2],ios::in);
  // Controllo apertura secondo file
  if(file_b.fail()) {
    cerr << "Errore apertura secondo file di input\n";
    exit(-1);
  }
  // Numero righe del secondo file
  file_b >> maxnum_b;
  // Apro file di output
  file_output.open(argv[3],ios::out);

  // Prima parola del primo file
  file_a >> parola_a;
  // Prima parola del secondo file
  file_b >> parola_b;
  for(int indice_a = 0, indice_b = 0;
      indice_a < maxnum_a;) {
    // Confronto tra le parole lette nei due file
    int test = strcmp(parola_a, parola_b);
    // Verifica risultato test
```

```
    if(test == 0) {
      // Sono uguali; allora la parola non va salvata,
      // vanno solo letti entrambi i file
      file_a >> parola_a;
      file_b >> parola_b;
      // E aggiornati entrambi gli indici
      indice_a++;
      indice_b++;
    } else if(test > 0 || indice_b >= maxnum_b) {
      // "parola_a" segue "parola_b", oppure
      // "file_b" e' terminato; allora
      // salvo la parola nel file di output
      file_output << parola_a << endl;
      // E leggo dal primo file
      file_a >> parola_a;
      // E aggiorno l'indice
      indice_a++;
    } else {
      // "parola_b" segue "parola_a"; allora
      // leggo dal secondo file
      file_b >> parola_b;
      // E aggiorno l'indice
      indice_b++;
    }
  }

  // Chiusura stream primo file
  file_a.close();
  // Chiusura stream secondo file
  file_b.close();
  // Chiusura stream file di output
  file_output.close();

  return(0);
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `shift` che, dato un vettore di interi `v` di dimensione `N` ed un intero positivo $J \leq N$, ritorni un nuovo vettore ottenuto dallo spostamento a sinistra di J posizioni del contenuto del vettore `v` e dall'inserimento di uno zero nelle posizioni lasciate libere.

Per esempio, dato il vettore $v = [2, 17, 44, 202, 5, 13]$ e l'intero $J = 3$, la funzione `shift` deve ritornare il vettore $[202, 5, 13, 0, 0, 0]$.

NOTA 1: **La funzione `shift` deve essere ricorsiva: al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

// Inserire qui sotto la dichiarazione della funzione shift

int* shift(int v[], int n, int j);
void left_shift(int v1[], int v2[], int n, int j, int i);

int main(){

   int J = 0;
   const int N = 15;
   int vector[] = {2, 17, 44, 202, 5, 13, 26, 7, 9, 131, 51, 79, 88, 96, 32};

   cout << "Array iniziale: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   cout << "Numero spostamenti a sinistra: ";
   cin >> J;

   int* new_vect = shift(vector, N, J);

   cout << "Nuovo array: ";
   for(int i=0; i<N; i++) {
      cout << new_vect[i] << " ";
   }
   cout << endl;

   delete[] new_vect;

return 0;
}

// Inserire qui sotto la definizione della funzione shift


int* shift(int v[], int n, int j) {
   int *new_v = new int[n];
   left_shift(v, new_v, n, j, 0);
   return new_v;
}

void left_shift(int v1[], int v2[], int n, int j, int i) {
   if (i >= n) {
      return;
   } else {
      if (i+j < n) {
         v2[i] = v1[i+j];
      } else {
```

```
            v2[i] = 0;
        }
        left_shift(v1, v2, n, j, i+1);
    }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `shift` che, dato un vettore di float `v` di dimensione `N` ed un intero positivo $J \leq N$, ritorni un nuovo vettore ottenuto dallo spostamento a destra di J posizioni del contenuto del vettore `v` e dall'inserimento di uno zero nelle posizioni lasciate libere.

Per esempio, dato il vettore $v = [0.3, 5.5, 13.2, 21.2, 7.9]$ e l'intero $J = 2$, la funzione `shift` deve ritornare il vettore $[0.0, 0.0, 0.3, 5.5, 13.2]$.

NOTA 1: **La funzione `shift` deve essere ricorsiva: al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

// Inserire qui sotto la dichiarazione della funzione shift

float* shift(float v[], int n, int j);
void right_shift(float v1[], float v2[], int n, int j, int i);

int main(){

   int J = 0;
   const int N = 10;
   float vector[] = {5.0, 13.2, 26.5, 7.2, 10.7, 9.3, 14.0, 81.0, 65.9, 32.1};

   cout << "Array iniziale: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   cout << "Numero spostamenti a destra: ";
   cin >> J;

   float* new_vect = shift(vector, N, J);

   cout << "Nuovo array: ";
   for(int i=0; i<N; i++) {
      cout << new_vect[i] << " ";
   }
   cout << endl;

   delete[] new_vect;

return 0;
}

// Inserire qui sotto la definizione della funzione shift


float* shift(float v[], int n, int j) {
   float *new_v = new float[n];
   right_shift(v, new_v, n, j, 0);
   return new_v;
}

void right_shift(float v1[], float v2[], int n, int j, int i) {
   if (i >= n) {
      return;
   } else {
      if (i < j) {
         v2[i] = 0.0;
      } else {
```

```
            v2[i] = v1[i-j];
        }
        right_shift(v1, v2, n, j, i+1);
    }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `shift` che, dato un vettore di caratteri minuscoli `v` di dimensione `N` ed un intero positivo $J \leq N$, ritorni un nuovo vettore ottenuto dallo spostamento a destra di J posizioni del contenuto del vettore `v` e dall'inserimento di del carattere `'x'` nelle posizioni lasciate libere.

Per esempio, dato il vettore $v = [d, c, p, m, z, l]$ e l'intero $J = 2$, la funzione `shift` deve ritornare il vettore $[x, x, d, c, p, m]$.

NOTA 1: **La funzione `shift` deve essere ricorsiva: al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

// Inserire qui sotto la dichiarazione della funzione shift

char* shift(char v[], int n, int j);
void right_shift(char v1[], char v2[], int n, int j, int i);

int main(){

   int J = 0;
   const int N = 12;
   char vector[] = {'c', 'l', 'w', 'a', 'p', 'r', 't', 'm', 'q', 'e', 'y', 'v'};

   cout << "Array iniziale: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   cout << "Numero spostamenti a destra: ";
   cin >> J;

   char* new_vect = shift(vector, N, J);

   cout << "Nuovo array: ";
   for(int i=0; i<N; i++) {
      cout << new_vect[i] << " ";
   }
   cout << endl;

   delete[] new_vect;

return 0;
}

// Inserire qui sotto la definizione della funzione shift


char* shift(char v[], int n, int j) {
   char *new_v = new char[n];
   right_shift(v, new_v, n, j, 0);
   return new_v;
}

void right_shift(char v1[], char v2[], int n, int j, int i) {
   if (i >= n) {
      return;
   } else {
      if (i < j) {
         v2[i] = 'x';
      } else {
```

```
            v2[i] = v1[i-j];
        }
        right_shift(v1, v2, n, j, i+1);
    }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `shift` che, dato un vettore di caratteri maiuscoli `v` di dimensione `N` ed un intero positivo $J \leq N$, ritorni un nuovo vettore ottenuto dallo spostamento a sinistra di J posizioni del contenuto del vettore `v` e dall'inserimento di del carattere `'Y'` nelle posizioni lasciate libere.

Per esempio, dato il vettore $v = [L, P, D, A, R, I]$ e l'intero $J = 4$, la funzione `shift` deve ritornare il vettore $[R, I, Y, Y, Y, Y]$.

NOTA 1: **La funzione `shift` deve essere ricorsiva: al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

// Inserire qui sotto la dichiarazione della funzione shift

char* shift(char v[], int n, int j);
void left_shift(char v1[], char v2[], int n, int j, int i);

int main(){

   int J = 0;
   const int N = 10;
   char vector[] = {'A', 'Z', 'E', 'I', 'K', 'P', 'N', 'F', 'D', 'J'};

   cout << "Array iniziale: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   cout << "Numero spostamenti a sinistra: ";
   cin >> J;

   char* new_vect = shift(vector, N, J);

   cout << "Nuovo array: ";
   for(int i=0; i<N; i++) {
      cout << new_vect[i] << " ";
   }
   cout << endl;

   delete[] new_vect;

return 0;
}

// Inserire qui sotto la definizione della funzione shift


char* shift(char v[], int n, int j) {
   char *new_v = new char[n];
   left_shift(v, new_v, n, j, 0);
   return new_v;
}

void left_shift(char v1[], char v2[], int n, int j, int i) {
   if (i >= n) {
      return;
   } else {
      if (i+j < n) {
         v2[i] = v1[i+j];
      } else {
```

```
            v2[i] = 'Y';
        }
        left_shift(v1, v2, n, j, i+1);
    }
}
```

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

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

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

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

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

3 **stack_main.cc**

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

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    int val;

    init(s);

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

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

    deinit(s);

    return 0;
}
```

3 stack.h

```
#ifndef STACK_H
#define STACK_H

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

typedef node *stack;

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

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

#endif // STACK_H
```

3 soluzione_A31.cc

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

#include "stack.h"


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

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

```
    s = NULL;
}

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

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

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

retval pop (stack &s)
{
    retval res = FALSE;
    if (!empty(s)) {
      node *first = s;
      s = s->next;
      delete first;

      res = TRUE;
    }
    return res;
}

retval top (const stack &s, int &result)
{
    retval res = FALSE;
    if (!empty(s)) {
      result = s->val;
      res = TRUE;
    }
    return res;
}

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

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

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

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

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

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

3 **stack_main.cc**

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

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    long val;

    init(s);

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

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

    deinit(s);

    return 0;
}
```

3 stack.h

```
#ifndef STACK_H
#define STACK_H

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

typedef node *stack;

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

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

#endif // STACK_H
```

3 soluzione_A32.cc

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

#include "stack.h"


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

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

```
    s = NULL;
}

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

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

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

retval shrink (stack &s)
{
    retval res = FALSE;
    if (nempty(s)) {
      node *first = s;
      s = s->next;
      delete first;

      res = TRUE;
    }
    return res;
}

retval first (const stack &s, long &result)
{
    retval res = FALSE;
    if (nempty(s)) {
      result = s->val;
      res = TRUE;
    }
    return res;
}

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

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

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

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

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

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

3 **stack_main.cc**

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

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    char val;

    init(s);

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

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

    deinit(s);

    return 0;
}
```

3 stack.h

```
#ifndef STACK_H
#define STACK_H

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

typedef node *stack;

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

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

#endif // STACK_H
```

3 soluzione_A33.cc

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

#include "stack.h"


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

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

```
    s = NULL;
}

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

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

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

retval pop (stack &s)
{
    retval res = FALSE;
    if (nempty(s)) {
      node *first = s;
      s = s->next;
      delete first;

      res = TRUE;
    }
    return res;
}

retval top (const stack &s, char &result)
{
    retval res = FALSE;
    if (nempty(s)) {
      result = s->val;
      res = TRUE;
    }
    return res;
}

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

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

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

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

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

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

3 **stack_main.cc**

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

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    double val;

    init(s);

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

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

    deinit(s);

    return 0;
}
```

3 stack.h

```
#ifndef STACK_H
#define STACK_H

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

typedef node *stack;

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

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

#endif // STACK_H
```

3 soluzione_A34.cc

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

#include "stack.h"


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

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

```
    s = NULL;
}

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

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

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

retval shrink (stack &s)
{
    retval res = FALSE;
    if (!empty(s)) {
      node *first = s;
      s = s->next;
      delete first;

      res = TRUE;
    }
    return res;
}

retval first (const stack &s, double &result)
{
    retval res = FALSE;
    if (!empty(s)) {
      result = s->val;
      res = TRUE;
    }
    return res;
}

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

4 Si scriva l'esercizio 1 *senza utilizzare l'allocazione dinamica* e *in forma RICORSIVA*; deve cioè essere implementato tramite una funzione ricorsiva, al cui interno non ci possono essere cicli o chiamate a funzioni contenenti cicli; è comunque ammesso l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

Se si fosse già risolto l'esercizio 1 in forma ricorsiva, se corretto la soluzione verrà valutata valida per la lode.

NOTA: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria, con gli stessi vincoli già identificati per la risoluzione dell'esercizio 1.

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

4 **esercizio4.cc**

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

using namespace std;

const int MAX_CHAR = 255 + 1;

void leggi_e_confronta_file(fstream&, fstream&, fstream&,
                            int, int, int, int, char[], char[]);

int main(int argc,char* argv[]){
  fstream file_a, file_b, file_output;
  int maxnum_a, maxnum_b;
  char parola_a[MAX_CHAR], parola_b[MAX_CHAR];

  // Controllo parametri in ingresso
  if (argc != 4) {
    cerr << "Sintassi: ./a.out <file_a> <file_b> <risultato>\n";
    exit(-1);
  }

  // Apro il primo file
  file_a.open(argv[1],ios::in);
  // Controllo apertura primo file
  if(file_a.fail()) {
    cerr << "Errore apertura primo file di input\n";
    exit(-1);
  }
  // Numero righe del primo file
  file_a >> maxnum_a;
  // Apro il secondo file
  file_b.open(argv[2],ios::in);
  // Controllo apertura secondo file
  if(file_b.fail()) {
    cerr << "Errore apertura secondo file di input\n";
    exit(-1);
  }
  // Numero righe del secondo file
  file_b >> maxnum_b;
  // Apro file di output
  file_output.open(argv[3],ios::out);

  // Prima parola del primo file
  file_a >> parola_a;
  // Prima parola del secondo file
  file_b >> parola_b;
  // Lettura simultanea dei due file
  leggi_e_confronta_file(file_a, file_b, file_output, 0, maxnum_a,
                         0, maxnum_b, parola_a, parola_b);
```

```
  // Chiusura stream primo file
  file_a.close();
  // Chiusura stream secondo file
  file_b.close();
  // Chiusura stream file di output
  file_output.close();

  return(0);
}

void leggi_e_confronta_file(fstream& file_a, fstream& file_b, fstream& file_output,
                            int indice_a, int maxnum_a, int indice_b, int maxnum_b,
                            char parola_a[], char parola_b[]) {
  if(indice_a < maxnum_a && indice_b < maxnum_b) {
    // Confronto tra le parole lette nei due file
    int test = strcmp(parola_a, parola_b);
    // Verifica risultato test
    if(test == 0) {
      // Sono uguali; allora la parola va salvata
      file_output << parola_a << endl;
      // E vanno letti entrambi i file
      file_a >> parola_a;
      file_b >> parola_b;
      // E aggiornati entrambi gli indici
      indice_a++;
      indice_b++;
    } else if(test < 0) {
      // "parola_a" precede "parola_b"; allora
      // leggo dal primo file
      file_a >> parola_a;
      // E aggiorno l'indice
      indice_a++;
    } else {
      // "parola_b" precede "parola_a"; allora
      // leggo dal secondo file
      file_b >> parola_b;
      // E aggiorno l'indice
      indice_b++;
    }
    // Passo ricorsivo
    leggi_e_confronta_file(file_a, file_b, file_output, indice_a, maxnum_a,
                            indice_b, maxnum_b, parola_a, parola_b);
  }
}
```