Quarto Appello di Programmazione I

09 luglio 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 Si leggano da un file di testo tutte le parole presenti, scartando quelle che precedono la parola "START"; nel caso quest'ultima non sia presente, vanno scartate tutte le parole del file di ingresso. Le parole così lette vanno poi salvate in un file di output in ordine **inverso** rispetto a come vengono lette.
Una parola nel file di ingresso corrisponde ad una sequenza di caratteri separata dalla successiva da uno spazio o da un "a capo": nel file di output le parole sono separate tra loro da uno spazio.

Scrivere nel file `esercizio1.cc` un programma che, invocato con la seguente sintassi:

```
./a.out <file_input> <file_output>
```

generi il contenuto del file `file_output` come descritto sopra. La parola chiave "START" **non** va inclusa nel file di output.

Si assuma per semplicità che il file di ingresso contenga parole di lunghezza non superiore a 50 caratteri.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation START ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figura 1: `primo esempio`

Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura, ché la diritta via era smarrita. Ahi quanto a dir qual era è cosa dura, esta selva selvaggia e aspra e forte, che nel pensier rinova la paura! Tant'è amara che poco è più morte; ma per trattar del ben ch'i' vi trovai, dirò de l'altre cose ch'i' v'ho scorte. Io non so ben ridir com'i' v'intrai, tant'era pien di sonno a quel punto che la verace via abbandonai.
Dante Alighieri, Inferno, I, vv. 1-12

Figura 2: `secondo esempio`

Dato in ingresso il file di input di cui alla Figura 1, si ottiene come risultato:

```
exercitation nostrud quis veniam, minim ad enim Ut aliqua. magna dolore et labore
ut incididunt tempor eiusmod do sed elit, adipiscing consectetur amet, sit dolor
ipsum Lorem
```

Mentre, dato in ingresso il file di input di cui alla Figura 2, si ottiene come risultato un file vuoto.

NOTA 1: Il file di input va scorso dalla prima all'ultima parola, in sequenza; è ammesso leggere il file di input una oppure due volte, se ritenuto necessario.

NOTA 2: È possibile utilizzare tutte le funzioni della libreria `cstring`; **non e' ammesso l'utilizzo di altre funzioni di libreria**.

NOTA 3: Va implementato il controllo dei parametri in ingresso, in particolare il file di input deve esistere ed essere accessibile in lettura.

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

1 **esercizio1.cc**

```
// Soluzione 1 (iterativa, con allocazione dinamica e due letture)

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

using namespace std;

const int DIM_PAROLA = 50 + 1;
const char* PAROLA_START = "START";

int main(int argc, char * argv[]) {
  fstream file_input, file_output;

  // Controllo argomenti passati in ingresso
  if (argc != 3) {
      cerr << "Sintassi: ./a.out <file_di_input> <file_di_output>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[1], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[1] << ")\n";
      exit(EXIT_FAILURE);
  }

  // Buffer per una parola
  char parola[DIM_PAROLA];
  // Contatore parole del file
  int numero_parole = 0;
  // Leggo la prima parola del file
  file_input >> parola;
  // Primo semi-ciclo di lettura, per raggiungere il punto in cui
  // e' presente la "PAROLA_START"
  while(!file_input.eof() && (strcmp(parola, PAROLA_START) != 0)) {
    file_input >> parola;
  }
  // Leggo la prima parola dopo la "PAROLA_START"
  file_input >> parola;
  // Secondo semi-ciclo di lettura, per stimare la dimensione
  // residua del file
  while(!file_input.eof()) {
    numero_parole++;
    file_input >> parola;
  }

  // Chiudo e riapro il file
  file_input.close();
  file_input.open(argv[1], ios::in);
```

```
  // Alloco lo spazio per salvare le parole in memoria
  char** parole = new char* [numero_parole];
  // Terzo semi-ciclo di lettura, per raggiungere il punto in cui
  // e' presente la "PAROLA_START"
  while(!file_input.eof() && (strcmp(parola, PAROLA_START) != 0)) {
    file_input >> parola;
  }
  // Quarto semi-ciclo di lettura, per salvare il contenuto in memoria
  for(int i = 0; i < numero_parole; i++) {
    file_input >> parola;
    // Alloco lo spazio per ciascuna parola
    parole[i] = new char[strlen(parola)];
    strcpy(parole[i], parola);
  }
  // Chiude il file di input
  file_input.close();

  // Apertura file di output
  file_output.open(argv[2], ios::out);
  // Salvo le parole sul secondo file, in ordine inverso
  for(int i = numero_parole - 1; i >= 0; i--) {
    file_output << parole[i] << " ";
  }
  // Chiude il file di output
  file_output.close();

  return 0;
}
```

```
// Soluzione 2 (ricorsiva)

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

using namespace std;

const int DIM_PAROLA = 50 + 1;
const char* PAROLA_START = "START";

void leggi_e_stampa(fstream& in, fstream& out);
void stampa_inversa(fstream& in, fstream& out);

int main(int argc, char * argv[]) {
  fstream file_input, file_output;

  // Controllo argomenti passati in ingresso
  if (argc != 3) {
      cerr << "Sintassi: ./a.out <file_di_input> <file_di_output>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[1], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[1] << ")\n";
      exit(EXIT_FAILURE);
  }

  // Apertura file di output
  file_output.open(argv[2], ios::out);

  // Chiamata ricorsiva
  leggi_e_stampa(file_input, file_output);

  // Chiude i file
  file_input.close();
  file_output.close();

  return 0;
}

void leggi_e_stampa(fstream& in, fstream& out) {
  // Escludo il caso base
  if(!in.eof()) {
    // Buffer per una parola
    char parola[DIM_PAROLA];
    // Leggo una parola del file
    in >> parola;
    // Test sulla parola di start
    if(strcmp(parola, PAROLA_START) != 0) {
      // Chiamata ricorsiva
```

```
      leggi_e_stampa(in, out);
    } else {
      // Chiama la funzione ricorsiva che stampa
      // tutte le parole da questo punto alla fine
      // del file
      stampa_inversa(in, out);
    }
  }
}

void stampa_inversa(fstream& in, fstream& out) {
  // Buffer per una parola
  char parola[DIM_PAROLA];
  // Leggo una parola del file
  in >> parola;
  // Caso base, file finito
  if(!in.eof()) {
    // Chiamata ricorsiva
    stampa_inversa(in, out);
    // Salvo la parola sul file di output
    out << parola << " ";
  }
}
```

1 Si leggano da un file di testo tutte le parole presenti, scartando quelle che precedono la parola "INIZIO"; nel caso quest'ultima non sia presente, vanno scartate tutte le parole del file di ingresso. Le parole così lette vanno poi salvate in un file di output in ordine **inverso** rispetto a come vengono lette.
Una parola nel file di ingresso corrisponde ad una sequenza di caratteri separata dalla successiva da uno spazio o da un "a capo": nel file di output le parole sono separate tra loro da uno spazio.

Scrivere nel file `esercizio1.cc` un programma che, invocato con la seguente sintassi:

```
./a.out <file_input> <file_output>
```

generi il contenuto del file `file_output` come descritto sopra. La parola chiave "INIZIO" **non** va inclusa nel file di output.

Si assuma per semplicità che il file di ingresso contenga parole di lunghezza non superiore a 50 caratteri.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit INIZIO in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figura 3: `primo esempio`

Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura, ché la diritta via era smarrita. Ahi quanto a dir qual era è cosa dura, esta selva selvaggia e aspra e forte, che nel pensier rinova la paura! Tant'è amara che poco è più morte; ma per trattar del ben ch'i' vi trovai, dirò de l'altre cose ch'i' v'ho scorte. Io non so ben ridir com'i' v'intrai, tant'era pien di sonno a quel punto che la verace via abbandonai. Dante Alighieri, Inferno, I, vv. 1-12

Figura 4: `secondo esempio`

Dato in ingresso il file di input di cui alla Figura 1, si ottiene come risultato:

```
laborum. est id anim mollit deserunt officia qui culpa in sunt proident,
non cupidatat occaecat sint Excepteur pariatur. nulla fugiat eu dolore
cillum esse velit voluptate in
```

Mentre, dato in ingresso il file di input di cui alla Figura 2, si ottiene come risultato un file vuoto.

NOTA 1: Il file di input va scorso dalla prima all'ultima parola, in sequenza; è ammesso leggere il file di input una oppure due volte, se ritenuto necessario.

NOTA 2: È possibile utilizzare tutte le funzioni della libreria `cstring`; **non e' ammesso l'utilizzo di altre funzioni di libreria**.

NOTA 3: Va implementato il controllo dei parametri in ingresso, in particolare il file di input deve esistere ed essere accessibile in lettura.

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

1 **esercizio1.cc**

```
// Soluzione 1 (iterativa, con allocazione dinamica e due letture)

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

using namespace std;

const int DIM_PAROLA = 50 + 1;
const char* PAROLA_INIZIO = "INIZIO";

int main(int argc, char * argv[]) {
  fstream file_input, file_output;

  // Controllo argomenti passati in ingresso
  if (argc != 3) {
      cerr << "Sintassi: ./a.out <file_di_input> <file_di_output>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[1], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[1] << ")\n";
      exit(EXIT_FAILURE);
  }

  // Buffer per una parola
  char parola[DIM_PAROLA];
  // Contatore parole del file
  int numero_parole = 0;
  // Leggo la prima parola del file
  file_input >> parola;
  // Primo semi-ciclo di lettura, per raggiungere il punto in cui
  // e' presente la "PAROLA_INIZIO"
  while(!file_input.eof() && (strcmp(parola, PAROLA_INIZIO) != 0)) {
    file_input >> parola;
  }
  // Leggo la prima parola dopo la "PAROLA_INIZIO"
  file_input >> parola;
  // Secondo semi-ciclo di lettura, per stimare la dimensione
  // residua del file
  while(!file_input.eof()) {
    numero_parole++;
    file_input >> parola;
  }

  // Chiudo e riapro il file
  file_input.close();
  file_input.open(argv[1], ios::in);
```

```
  // Alloco lo spazio per salvare le parole in memoria
  char** parole = new char* [numero_parole];
  // Terzo semi-ciclo di lettura, per raggiungere il punto in cui
  // e' presente la "PAROLA_INIZIO"
  while(!file_input.eof() && (strcmp(parola, PAROLA_INIZIO) != 0)) {
    file_input >> parola;
  }
  // Quarto semi-ciclo di lettura, per salvare il contenuto in memoria
  for(int i = 0; i < numero_parole; i++) {
    file_input >> parola;
    // Alloco lo spazio per ciascuna parola
    parole[i] = new char[strlen(parola)];
    strcpy(parole[i], parola);
  }
  // Chiude il file di input
  file_input.close();

  // Apertura file di output
  file_output.open(argv[2], ios::out);
  // Salvo le parole sul secondo file, in ordine inverso
  for(int i = numero_parole - 1; i >= 0; i--) {
    file_output << parole[i] << " ";
  }
  // Chiude il file di output
  file_output.close();

  return 0;
}
```

```
// Soluzione 2 (ricorsiva)

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

using namespace std;

const int DIM_PAROLA = 50 + 1;
const char* PAROLA_INIZIO = "INIZIO";

void leggi_e_stampa(fstream& in, fstream& out);
void stampa_inversa(fstream& in, fstream& out);

int main(int argc, char * argv[]) {
  fstream file_input, file_output;

  // Controllo argomenti passati in ingresso
  if (argc != 3) {
      cerr << "Sintassi: ./a.out <file_di_input> <file_di_output>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[1], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[1] << ")\n";
      exit(EXIT_FAILURE);
  }

  // Apertura file di output
  file_output.open(argv[2], ios::out);

  // Chiamata ricorsiva
  leggi_e_stampa(file_input, file_output);

  // Chiude i file
  file_input.close();
  file_output.close();

  return 0;
}

void leggi_e_stampa(fstream& in, fstream& out) {
  // Escludo il caso base
  if(!in.eof()) {
    // Buffer per una parola
    char parola[DIM_PAROLA];
    // Leggo una parola del file
    in >> parola;
    // Test sulla parola di start
    if(strcmp(parola, PAROLA_INIZIO) != 0) {
      // Chiamata ricorsiva
```

```
      leggi_e_stampa(in, out);
    } else {
      // Chiama la funzione ricorsiva che stampa
      // tutte le parole da questo punto alla fine
      // del file
      stampa_inversa(in, out);
    }
  }
}

void stampa_inversa(fstream& in, fstream& out) {
  // Buffer per una parola
  char parola[DIM_PAROLA];
  // Leggo una parola del file
  in >> parola;
  // Caso base, file finito
  if(!in.eof()) {
    // Chiamata ricorsiva
    stampa_inversa(in, out);
    // Salvo la parola sul file di output
    out << parola << " ";
  }
}
```

1 Si leggano da un file di testo tutte le parole presenti, scartando quelle che precedono la parola "BEGIN"; nel caso quest'ultima non sia presente, vanno scartate tutte le parole del file di ingresso. Le parole così lette vanno poi salvate in un file di output in ordine **inverso** rispetto a come vengono lette.
Una parola nel file di ingresso corrisponde ad una sequenza di caratteri separata dalla successiva da uno spazio o da un "a capo": nel file di output le parole sono separate tra loro da uno spazio.

Scrivere nel file `esercizio1.cc` un programma che, invocato con la seguente sintassi:

```
./a.out <file_input> <file_output>
```

generi il contenuto del file `file_output` come descritto sopra. La parola chiave "BEGIN" **non** va inclusa nel file di output.

Si assuma per semplicità che il file di ingresso contenga parole di lunghezza non superiore a 50 caratteri.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. BEGIN Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figura 5: `primo esempio`

Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura, ché la diritta via era smarrita. Ahi quanto a dir qual era è cosa dura, esta selva selvaggia e aspra e forte, che nel pensier rinova la paura! Tant'è amara che poco è più morte; ma per trattar del ben ch'i' vi trovai, dirò de l'altre cose ch'i' v'ho scorte. Io non so ben ridir com'i' v'intrai, tant'era pien di sonno a quel punto che la verace via abbandonai. Dante Alighieri, Inferno, I, vv. 1-12

Figura 6: `secondo esempio`

Dato in ingresso il file di input di cui alla Figura 1, si ottiene come risultato:

```
laborum. est id anim mollit deserunt officia qui culpa in sunt proident,
non cupidatat occaecat sint Excepteur pariatur. nulla fugiat eu
dolore cillum esse velit voluptate in reprehenderit in dolor irure aute Duis
```

Mentre, dato in ingresso il file di input di cui alla Figura 2, si ottiene come risultato un file vuoto.

NOTA 1: Il file di input va scorso dalla prima all'ultima parola, in sequenza; è ammesso leggere il file di input una oppure due volte, se ritenuto necessario.

NOTA 2: È possibile utilizzare tutte le funzioni della libreria `cstring`; **non e' ammesso l'utilizzo di altre funzioni di libreria**.

NOTA 3: Va implementato il controllo dei parametri in ingresso, in particolare il file di input deve esistere ed essere accessibile in lettura.

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

1 **esercizio1.cc**

```
// Soluzione 1 (iterativa, con allocazione dinamica e due letture)

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

using namespace std;

const int DIM_PAROLA = 50 + 1;
const char* PAROLA_BEGIN = "BEGIN";

int main(int argc, char * argv[]) {
  fstream file_input, file_output;

  // Controllo argomenti passati in ingresso
  if (argc != 3) {
      cerr << "Sintassi: ./a.out <file_di_input> <file_di_output>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[1], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[1] << ")\n";
      exit(EXIT_FAILURE);
  }

  // Buffer per una parola
  char parola[DIM_PAROLA];
  // Contatore parole del file
  int numero_parole = 0;
  // Leggo la prima parola del file
  file_input >> parola;
  // Primo semi-ciclo di lettura, per raggiungere il punto in cui
  // e' presente la "PAROLA_BEGIN"
  while(!file_input.eof() && (strcmp(parola, PAROLA_BEGIN) != 0)) {
    file_input >> parola;
  }
  // Leggo la prima parola dopo la "PAROLA_BEGIN"
  file_input >> parola;
  // Secondo semi-ciclo di lettura, per stimare la dimensione
  // residua del file
  while(!file_input.eof()) {
    numero_parole++;
    file_input >> parola;
  }

  // Chiudo e riapro il file
  file_input.close();
  file_input.open(argv[1], ios::in);
```

```
  // Alloco lo spazio per salvare le parole in memoria
  char** parole = new char* [numero_parole];
  // Terzo semi-ciclo di lettura, per raggiungere il punto in cui
  // e' presente la "PAROLA_BEGIN"
  while(!file_input.eof() && (strcmp(parola, PAROLA_BEGIN) != 0)) {
    file_input >> parola;
  }
  // Quarto semi-ciclo di lettura, per salvare il contenuto in memoria
  for(int i = 0; i < numero_parole; i++) {
    file_input >> parola;
    // Alloco lo spazio per ciascuna parola
    parole[i] = new char[strlen(parola)];
    strcpy(parole[i], parola);
  }
  // Chiude il file di input
  file_input.close();

  // Apertura file di output
  file_output.open(argv[2], ios::out);
  // Salvo le parole sul secondo file, in ordine inverso
  for(int i = numero_parole - 1; i >= 0; i--) {
    file_output << parole[i] << " ";
  }
  // Chiude il file di output
  file_output.close();

  return 0;
}
```

```
// Soluzione 2 (ricorsiva)

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

using namespace std;

const int DIM_PAROLA = 50 + 1;
const char* PAROLA_BEGIN = "BEGIN";

void leggi_e_stampa(fstream& in, fstream& out);
void stampa_inversa(fstream& in, fstream& out);

int main(int argc, char * argv[]) {
  fstream file_input, file_output;

  // Controllo argomenti passati in ingresso
  if (argc != 3) {
      cerr << "Sintassi: ./a.out <file_di_input> <file_di_output>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[1], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[1] << ")\n";
      exit(EXIT_FAILURE);
  }

  // Apertura file di output
  file_output.open(argv[2], ios::out);

  // Chiamata ricorsiva
  leggi_e_stampa(file_input, file_output);

  // Chiude i file
  file_input.close();
  file_output.close();

  return 0;
}

void leggi_e_stampa(fstream& in, fstream& out) {
  // Escludo il caso base
  if(!in.eof()) {
    // Buffer per una parola
    char parola[DIM_PAROLA];
    // Leggo una parola del file
    in >> parola;
    // Test sulla parola di start
    if(strcmp(parola, PAROLA_BEGIN) != 0) {
      // Chiamata ricorsiva
```

```
      leggi_e_stampa(in, out);
    } else {
      // Chiama la funzione ricorsiva che stampa
      // tutte le parole da questo punto alla fine
      // del file
      stampa_inversa(in, out);
    }
  }
}

void stampa_inversa(fstream& in, fstream& out) {
  // Buffer per una parola
  char parola[DIM_PAROLA];
  // Leggo una parola del file
  in >> parola;
  // Caso base, file finito
  if(!in.eof()) {
    // Chiamata ricorsiva
    stampa_inversa(in, out);
    // Salvo la parola sul file di output
    out << parola << " ";
  }
}
```

1 Si leggano da un file di testo tutte le parole presenti, scartando quelle che precedono la parola "NEXT"; nel caso quest'ultima non sia presente, vanno scartate tutte le parole del file di ingresso. Le parole così lette vanno poi salvate in un file di output in ordine **inverso** rispetto a come vengono lette.
Una parola nel file di ingresso corrisponde ad una sequenza di caratteri separata dalla successiva da uno spazio o da un "a capo": nel file di output le parole sono separate tra loro da uno spazio.

Scrivere nel file `esercizio1.cc` un programma che, invocato con la seguente sintassi:

```
./a.out <file_input> <file_output>
```

generi il contenuto del file `file_output` come descritto sopra. La parola chiave "NEXT" **non** va inclusa nel file di output.

Si assuma per semplicità che il file di ingresso contenga parole di lunghezza non superiore a 50 caratteri.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea NEXT commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figura 7: `primo esempio`

Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura, ché la diritta via era smarrita. Ahi quanto a dir qual era è cosa dura, esta selva selvaggia e aspra e forte, che nel pensier rinova la paura! Tant'è amara che poco è più morte; ma per trattar del ben ch'i' vi trovai, dirò de l'altre cose ch'i' v'ho scorte. Io non so ben ridir com'i' v'intrai, tant'era pien di sonno a quel punto che la verace via abbandonai. Dante Alighieri, Inferno, I, vv. 1-12

Figura 8: `secondo esempio`

Dato in ingresso il file di input di cui alla Figura 1, si ottiene come risultato:

```
laborum. est id anim mollit deserunt officia qui culpa in sunt proident,
non cupidatat occaecat sint Excepteur pariatur. nulla fugiat eu dolore cillum
esse velit voluptate in reprehenderit in dolor irure aute Duis consequat. commodo
```

Mentre, dato in ingresso il file di input di cui alla Figura 2, si ottiene come risultato un file vuoto.

NOTA 1: Il file di input va scorso dalla prima all'ultima parola, in sequenza; è ammesso leggere il file di input una oppure due volte, se ritenuto necessario.

NOTA 2: È possibile utilizzare tutte le funzioni della libreria `cstring`; **non e' ammesso l'utilizzo di altre funzioni di libreria**.

NOTA 3: Va implementato il controllo dei parametri in ingresso, in particolare il file di input deve esistere ed essere accessibile in lettura.

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

1 **esercizio1.cc**

```
// Soluzione 1 (iterativa, con allocazione dinamica e due letture)

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

using namespace std;

const int DIM_PAROLA = 50 + 1;
const char* PAROLA_NEXT = "NEXT";

int main(int argc, char * argv[]) {
  fstream file_input, file_output;

  // Controllo argomenti passati in ingresso
  if (argc != 3) {
      cerr << "Sintassi: ./a.out <file_di_input> <file_di_output>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[1], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[1] << ")\n";
      exit(EXIT_FAILURE);
  }

  // Buffer per una parola
  char parola[DIM_PAROLA];
  // Contatore parole del file
  int numero_parole = 0;
  // Leggo la prima parola del file
  file_input >> parola;
  // Primo semi-ciclo di lettura, per raggiungere il punto in cui
  // e' presente la "PAROLA_NEXT"
  while(!file_input.eof() && (strcmp(parola, PAROLA_NEXT) != 0)) {
    file_input >> parola;
  }
  // Leggo la prima parola dopo la "PAROLA_NEXT"
  file_input >> parola;
  // Secondo semi-ciclo di lettura, per stimare la dimensione
  // residua del file
  while(!file_input.eof()) {
    numero_parole++;
    file_input >> parola;
  }

  // Chiudo e riapro il file
  file_input.close();
  file_input.open(argv[1], ios::in);
```

```
  // Alloco lo spazio per salvare le parole in memoria
  char** parole = new char* [numero_parole];
  // Terzo semi-ciclo di lettura, per raggiungere il punto in cui
  // e' presente la "PAROLA_NEXT"
  while(!file_input.eof() && (strcmp(parola, PAROLA_NEXT) != 0)) {
    file_input >> parola;
  }
  // Quarto semi-ciclo di lettura, per salvare il contenuto in memoria
  for(int i = 0; i < numero_parole; i++) {
    file_input >> parola;
    // Alloco lo spazio per ciascuna parola
    parole[i] = new char[strlen(parola)];
    strcpy(parole[i], parola);
  }
  // Chiude il file di input
  file_input.close();

  // Apertura file di output
  file_output.open(argv[2], ios::out);
  // Salvo le parole sul secondo file, in ordine inverso
  for(int i = numero_parole - 1; i >= 0; i--) {
    file_output << parole[i] << " ";
  }
  // Chiude il file di output
  file_output.close();

  return 0;
}
```

```
// Soluzione 2 (ricorsiva)

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

using namespace std;

const int DIM_PAROLA = 50 + 1;
const char* PAROLA_NEXT = "NEXT";

void leggi_e_stampa(fstream& in, fstream& out);
void stampa_inversa(fstream& in, fstream& out);

int main(int argc, char * argv[]) {
  fstream file_input, file_output;

  // Controllo argomenti passati in ingresso
  if (argc != 3) {
      cerr << "Sintassi: ./a.out <file_di_input> <file_di_output>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[1], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[1] << ")\n";
      exit(EXIT_FAILURE);
  }

  // Apertura file di output
  file_output.open(argv[2], ios::out);

  // Chiamata ricorsiva
  leggi_e_stampa(file_input, file_output);

  // Chiude i file
  file_input.close();
  file_output.close();

  return 0;
}

void leggi_e_stampa(fstream& in, fstream& out) {
  // Escludo il caso base
  if(!in.eof()) {
    // Buffer per una parola
    char parola[DIM_PAROLA];
    // Leggo una parola del file
    in >> parola;
    // Test sulla parola di start
    if(strcmp(parola, PAROLA_NEXT) != 0) {
      // Chiamata ricorsiva
```

```
      leggi_e_stampa(in, out);
    } else {
      // Chiama la funzione ricorsiva che stampa
      // tutte le parole da questo punto alla fine
      // del file
      stampa_inversa(in, out);
    }
  }
}

void stampa_inversa(fstream& in, fstream& out) {
  // Buffer per una parola
  char parola[DIM_PAROLA];
  // Leggo una parola del file
  in >> parola;
  // Caso base, file finito
  if(!in.eof()) {
    // Chiamata ricorsiva
    stampa_inversa(in, out);
    // Salvo la parola sul file di output
    out << parola << " ";
  }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `crea_vettore` che, dato un intero positivo `N`, crei e restituisca un nuovo vettore contenente i primi `N` numeri della **sequenza di Fibonacci**.

(Per definizione,

$F(1) = 1$
$F(2) = 1$
$F(n) = F(n-1) + F(n-2)$

per ogni $n > 0$).

Per esempio, i primi 6 numeri della sequenza sono

$1, 1, 2, 3, 5, 8$

cioè rispettivamente

$F(1), F(2), F(3), F(4), F(5), F(6)$

NOTA 1: **La funzione `crea_vettore` deve essere ricorsiva ed 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: Si ricorda che **non è ammesso** modificare in alcun modo la funzione main, in particolare la chiamata alla funzione crea_vettore, pena l'annullamento della prova.

NOTA 4: 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 crea_vettore

int* crea_vettore(int n);
void fibonacci(int v[], int n, int i);

int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;

   if (N <= 0) {
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vettore = crea_vettore(N);

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

   delete[] vettore;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

int* crea_vettore(int n) {
   int *v = new int[n];
   fibonacci(v, n, 0);
   return v;
}

void fibonacci(int v[], int n, int i) {
  // Caso base
  if(i < n) {
    switch(i) {
      case 0: case 1:
        v[i] = 1;
        break;
      default:
        v[i] = v[i - 1] + v[i - 2];
    }
    // Chiamata ricorsiva
```

```
    fibonacci(v, n, i + 1);
  }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `crea_vettore` che, dato un intero positivo `N`, crei e restituisca un nuovo vettore contenente i primi `N` numeri della **sequenza di Lucas**.

(Per definizione,

$L(1) = 1$
$L(2) = 3$
$L(n) = L(n-1) + L(n-2)$

per ogni $n > 0$).

Per esempio, i primi 6 numeri della sequenza sono

$1, 3, 4, 7, 11, 18$

cioè rispettivamente

$L(1), L(2), L(3), L(4), L(5), L(6)$

NOTA 1: **La funzione `crea_vettore` deve essere ricorsiva ed 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: Si ricorda che **non è ammesso** modificare in alcun modo la funzione main, in particolare la chiamata alla funzione crea_vettore, pena l'annullamento della prova.

NOTA 4: 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 crea_vettore

int* crea_vettore(int n);
void lucas(int v[], int n, int i);

int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;

   if (N <= 0) {
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vettore = crea_vettore(N);

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

   delete[] vettore;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

int* crea_vettore(int n) {
   int *v = new int[n];
   lucas(v, n, 0);
   return v;
}

void lucas(int v[], int n, int i) {
  // Caso base
  if(i < n) {
    switch(i) {
      case 0:
        v[i] = 1;
        break;
      case 1:
        v[i] = 3;
        break;
      default:
```

```
        v[i] = v[i - 1] + v[i - 2];
    }
    // Chiamata ricorsiva
    lucas(v, n, i + 1);
  }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `crea_vettore` che, dato un intero positivo `N`, crei e restituisca un nuovo vettore contenente i primi `N` numeri della **sequenza di Perrin**.

(Per definizione,

$P(0) = 3$
$P(1) = 0$
$P(2) = 2$
$P(n) = P(n-2) + P(n-3)$

per ogni $n > 0$).

Per esempio, i primi 6 numeri della sequenza sono

$3, 0, 2, 3, 2, 5$

cioè rispettivamente

$P(0), P(1), P(2), P(3), P(4), P(5)$

NOTA 1: **La funzione `crea_vettore` deve essere ricorsiva ed 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: Si ricorda che **non è ammesso** modificare in alcun modo la funzione main, in particolare la chiamata alla funzione crea_vettore, pena l'annullamento della prova.

NOTA 4: 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 crea_vettore

int* crea_vettore(int n);
void perrin(int v[], int n, int i);

int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;

   if (N <= 0) {
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vettore = crea_vettore(N);

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

   delete[] vettore;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

int* crea_vettore(int n) {
   int *v = new int[n];
   perrin(v, n, 0);
   return v;
}

void perrin(int v[], int n, int i) {
  // Caso base
  if(i < n) {
    switch(i) {
      case 0:
        v[i] = 3;
        break;
      case 1:
        v[i] = 0;
        break;
      case 2:
```

```
        v[i] = 2;
        break;
      default:
        v[i] = v[i - 2] + v[i - 3];
    }
    // Chiamata ricorsiva
    perrin(v, n, i + 1);
  }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `crea_vettore` che, dato un intero positivo `N`, crei e restituisca un nuovo vettore contenente i primi `N` numeri della **sequenza di Padovan**.

(Per definizione,

$P(0) = 1$
$P(1) = 1$
$P(2) = 1$
$P(n) = P(n-2) + P(n-3)$

per ogni $n > 0$).

Per esempio, i primi 6 numeri della sequenza sono

$1, 1, 1, 2, 2, 3$

cioè rispettivamente

$P(0), P(1), P(2), P(3), P(4), P(5)$

NOTA 1: **La funzione `crea_vettore` deve essere ricorsiva ed 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: Si ricorda che **non è ammesso** modificare in alcun modo la funzione main, in particolare la chiamata alla funzione crea_vettore, pena l'annullamento della prova.

NOTA 4: 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 crea_vettore

int* crea_vettore(int n);
void padovan(int v[], int n, int i);

int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;

   if (N <= 0) {
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vettore = crea_vettore(N);

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

   delete[] vettore;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

int* crea_vettore(int n) {
   int *v = new int[n];
   padovan(v, n, 0);
   return v;
}

void padovan(int v[], int n, int i) {
  // Caso base
  if(i < n) {
    switch(i) {
      case 0: case 1: case 2:
        v[i] = 1;
        break;
      default:
        v[i] = v[i - 2] + v[i - 3];
    }
    // Chiamata ricorsiva
```

```
    padovan(v, n, i + 1);
  }
}
```

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

- `leggi_coppia` legga una coppia di valori da standard input e ritorni una struct "coppia" con questi ultimi;
- `stampa_coppia` stampi la coppia di valori ricevuta in ingresso e il valore della loro `norma`;
- `compara` che, presa in ingresso due coppie, ritorni zero ("0") se le `norme` delle due coppie sono uguali, un numero minore di zero se la norma della prima coppia è inferiore alla norma della seconda, un numero maggiore di zero altrimenti. (definiamo norma di una coppia come **la radice quadrata della somma dei quadrati dei due membri della coppia**; in simboli, $\sqrt{x^2 + y^2}$);
- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **crescente** e se l'elemento è già presente deve essere inserito a **sinistra**. Esempio: l'inserimento dei seguenti valori:

  $(5, 4) \quad (4, 5) \quad (10, 7) \quad (12, 9)$

  deve produrre il seguente albero:

  $(10, 7)$

  $(4, 5) \qquad\qquad (12, 9)$

  $(5, 4)$

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

  $(4, 5[\mathtt{R} = 6.40312]) \quad (5, 4[\mathtt{R} = 6.40312]) \quad (10, 7[\mathtt{R} = 12.2066]) \quad (12, 9[\mathtt{R} = 15])$

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

3 albero_main.cc

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

using namespace std;

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

  return 0;
```

```
}
```

3 albero.h

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

struct coppia {
    int x;
    int y;
};

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

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

coppia leggi_coppia();
void stampa_coppia(coppia val);
int compara(coppia a, coppia b);
void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, coppia val);
boolean cerca(const Albero &t, coppia val);
void stampa(const Albero &t);

#endif
```

3 albero.cc

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

using namespace std;

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

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

coppia leggi_coppia() {
  coppia ris;
  cout << "\tx = ";
```

```
  cin >> ris.x;
  cout << "\ty = ";
  cin >> ris.y;
  return ris;
}

static double norma(coppia val) {
  return sqrt(val.x * val.x + val.y * val.y);
}

void stampa_coppia(coppia val) {
  cout << "(" << val.x << ", " << val.y <<
    " [R = " << norma(val) << "])";
}

int compara(coppia a, coppia b) {
  int ris = 0;
  double norma_a, norma_b;
  norma_a = norma(a);
  norma_b = norma(b);
  if(norma_a > norma_b) {
    ris = 1;
  } else if (norma_b > norma_a) {
    ris = -1;
  }
  return ris;
}

boolean inserisci(Albero &a, coppia val) {
  // Caso base
  boolean res = FALSO;
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a != NULL) {
      a->val = val;
      a->sx = a->dx = NULL;
      res = VERO;
    }
  } else if (compara(val, a->val) <= 0) {
    // Caso ricorsivo: scendo a sinistra
    res = inserisci(a->sx, val);
  } else {
    // Caso ricorsivo: scendo a destra
    res = inserisci(a->dx, val);
  }
  return res;
}

boolean cerca(const Albero &a, coppia val) {
  boolean res = FALSO;
  if (vuoto(a) == FALSO) {
    int c = compara(val, a -> val);
    if (c == 0) {
      // Trovato
```

```
      res = VERO;
    } else if (c < 0) {
      // Scendo a sinistra
      res = cerca(a->sx, val);
    } else {
      // Scendo a destra
      res = cerca(a->dx, val);
    }
  }
  return res;
}

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

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

- `leggi_coppia` legga una coppia di valori da standard input e ritorni una struct "coppia" con questi ultimi;
- `stampa_coppia` stampi la coppia di valori ricevuta in ingresso e il valore della loro `norma`;
- `compara` che, presa in ingresso due coppie, ritorni zero ("0") se le `norme` delle due coppie sono uguali, un numero minore di zero se la norma della prima coppia è inferiore alla norma della seconda, un numero maggiore di zero altrimenti. (definiamo norma di una coppia come **la radice quadrata della somma dei quadrati dei due membri della coppia**; in simboli, $\sqrt{x^2 + y^2}$);
- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **decrescente** e se l'elemento è già presente deve essere inserito a **sinistra**. Esempio: l'inserimento dei seguenti valori:

  $$(9,7) \quad (12,3) \quad (5,4) \quad (-3,-12)$$

  deve produrre il seguente albero:

  $$\begin{array}{ccc} & (9,7) & \\ (12,3) & & (5,4) \\ (-3,-12) & & \end{array}$$

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

$$(5,4[\mathtt{R} = 6.40312]) \quad (9,7[\mathtt{R} = 11.4018]) \quad (12,3[\mathtt{R} = 12.3693]) \quad (-3,-12[\mathtt{R} = 12.3693])$$

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

3 **albero_main.cc**

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

using namespace std;

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

  return 0;
```

```
}
```

3 albero.h

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

struct coppia {
    int x;
    int y;
};

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

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

coppia leggi_coppia();
void stampa_coppia(coppia val);
int compara(coppia a, coppia b);
void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, coppia val);
boolean cerca(const Albero &t, coppia val);
void stampa(const Albero &t);

#endif
```

3 albero.cc

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

using namespace std;

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

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

coppia leggi_coppia() {
  coppia ris;
  cout << "\tx = ";
```

```
  cin >> ris.x;
  cout << "\ty = ";
  cin >> ris.y;
  return ris;
}

static double norma(coppia val) {
  return sqrt(val.x * val.x + val.y * val.y);
}

void stampa_coppia(coppia val) {
  cout << "(" << val.x << ", " << val.y <<
    " [R = " << norma(val) << "])";
}

int compara(coppia a, coppia b) {
  int ris = 0;
  double norma_a, norma_b;
  norma_a = norma(a);
  norma_b = norma(b);
  if(norma_a > norma_b) {
    ris = 1;
  } else if (norma_b > norma_a) {
    ris = -1;
  }
  return ris;
}

boolean inserisci(Albero &a, coppia val) {
  // Caso base
  boolean res = FALSO;
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a != NULL) {
      a->val = val;
      a->sx = a->dx = NULL;
      res = VERO;
    }
  } else if (compara(val, a->val) < 0) {
    // Caso ricorsivo: scendo a destra
    res = inserisci(a->dx, val);
  } else {
    // Caso ricorsivo: scendo a sinistra
    res = inserisci(a->sx, val);
  }
  return res;
}

boolean cerca(const Albero &a, coppia val) {
  boolean res = FALSO;
  if (vuoto(a) == FALSO) {
    int c = compara(val, a -> val);
    if (c == 0) {
      // Trovato
```

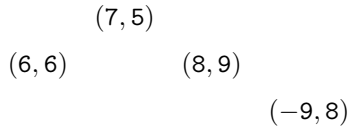
```
      res = VERO;
    } else if (c < 0) {
      // Scendo a destra
      res = cerca(a->dx, val);
    } else {
      // Scendo a sinistra
      res = cerca(a->sx, val);
    }
  }
  return res;
}

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

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

- `leggi_coppia` legga una coppia di valori da standard input e ritorni una struct "coppia" con questi ultimi;
- `stampa_coppia` stampi la coppia di valori ricevuta in ingresso e il valore della loro `norma`;
- `compara` che, presa in ingresso due coppie, ritorni zero ("0") se le `norme` delle due coppie sono uguali, un numero minore di zero se la norma della prima coppia è inferiore alla norma della seconda, un numero maggiore di zero altrimenti. (definiamo norma di una coppia come **la radice quadrata della somma dei quadrati dei due membri della coppia**; in simboli, $\sqrt{x^2 + y^2}$);
- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **crescente** e se l'elemento è già presente deve essere inserito a **destra**. Esempio: l'inserimento dei seguenti valori:

  $(7,5) \quad (8,9) \quad (6,6) \quad (-9,8)$

  deve produrre il seguente albero:

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

  $(6,6[\mathtt{R} = 8.48528]) \quad (7,5[\mathtt{R} = 8.60233]) \quad (8,9[\mathtt{R} = 12.0416]) \quad (-9,8[\mathtt{R} = 12.0416])$

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

3 **albero_main.cc**

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

using namespace std;

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

  return 0;
```

```
}
```

3 albero.h

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

struct coppia {
    int x;
    int y;
};

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

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

coppia leggi_coppia();
void stampa_coppia(coppia val);
int compara(coppia a, coppia b);
void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, coppia val);
boolean cerca(const Albero &t, coppia val);
void stampa(const Albero &t);

#endif
```

3 albero.cc

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

using namespace std;

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

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

coppia leggi_coppia() {
  coppia ris;
  cout << "\tx = ";
```

```
  cin >> ris.x;
  cout << "\ty = ";
  cin >> ris.y;
  return ris;
}

static double norma(coppia val) {
  return sqrt(val.x * val.x + val.y * val.y);
}

void stampa_coppia(coppia val) {
  cout << "(" << val.x << ", " << val.y <<
    " [R = " << norma(val) << "])";
}

int compara(coppia a, coppia b) {
  int ris = 0;
  double norma_a, norma_b;
  norma_a = norma(a);
  norma_b = norma(b);
  if(norma_a > norma_b) {
    ris = 1;
  } else if (norma_b > norma_a) {
    ris = -1;
  }
  return ris;
}

boolean inserisci(Albero &a, coppia val) {
  // Caso base
  boolean res = FALSO;
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a != NULL) {
      a->val = val;
      a->sx = a->dx = NULL;
      res = VERO;
    }
  } else if (compara(val, a->val) < 0) {
    // Caso ricorsivo: scendo a sinistra
    res = inserisci(a->sx, val);
  } else {
    // Caso ricorsivo: scendo a destra
    res = inserisci(a->dx, val);
  }
  return res;
}

boolean cerca(const Albero &a, coppia val) {
  boolean res = FALSO;
  if (vuoto(a) == FALSO) {
    int c = compara(val, a -> val);
    if (c == 0) {
      // Trovato
```

```
      res = VERO;
    } else if (c < 0) {
      // Scendo a sinistra
      res = cerca(a->sx, val);
    } else {
      // Scendo a destra
      res = cerca(a->dx, val);
    }
  }
  return res;
}

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

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

- `leggi_coppia` legga una coppia di valori da standard input e ritorni una struct "coppia" con questi ultimi;
- `stampa_coppia` stampi la coppia di valori ricevuta in ingresso e il valore della loro `norma`;
- `compara` che, presa in ingresso due coppie, ritorni zero ("0") se le `norme` delle due coppie sono uguali, un numero minore di zero se la norma della prima coppia è inferiore alla norma della seconda, un numero maggiore di zero altrimenti. (definiamo norma di una coppia come **la radice quadrata della somma dei quadrati dei due membri della coppia**; in simboli, $\sqrt{x^2 + y^2}$);
- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **decrescente** e se l'elemento è già presente deve essere inserito a **destra**. Esempio: l'inserimento dei seguenti valori:

  $(8, 2) \quad (9, 1) \quad (1, 5) \quad (1, -5)$

  deve produrre il seguente albero:

  $(8, 2)$

  $(9, 1) \qquad (1, 5)$

  $(1, -5)$

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

  $(1, -5[\mathtt{R} = 5.09902]) \quad (1, 5[\mathtt{R} = 5.09902]) \quad (8, 2[\mathtt{R} = 8.24621]) \quad (9, 1[\mathtt{R} = 9.05539])$

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

3 **albero_main.cc**

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

using namespace std;

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

  return 0;
```

```
}
```

3 albero.h

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

struct coppia {
    int x;
    int y;
};

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

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

coppia leggi_coppia();
void stampa_coppia(coppia val);
int compara(coppia a, coppia b);
void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, coppia val);
boolean cerca(const Albero &t, coppia val);
void stampa(const Albero &t);

#endif
```

3 albero.cc

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

using namespace std;

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

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

coppia leggi_coppia() {
  coppia ris;
  cout << "\tx = ";
```

```
  cin >> ris.x;
  cout << "\ty = ";
  cin >> ris.y;
  return ris;
}

static double norma(coppia val) {
  return sqrt(val.x * val.x + val.y * val.y);
}

void stampa_coppia(coppia val) {
  cout << "(" << val.x << ", " << val.y <<
    " [R = " << norma(val) << "])";
}

int compara(coppia a, coppia b) {
  int ris = 0;
  double norma_a, norma_b;
  norma_a = norma(a);
  norma_b = norma(b);
  if(norma_a > norma_b) {
    ris = 1;
  } else if (norma_b > norma_a) {
    ris = -1;
  }
  return ris;
}

boolean inserisci(Albero &a, coppia val) {
  // Caso base
  boolean res = FALSO;
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a != NULL) {
      a->val = val;
      a->sx = a->dx = NULL;
      res = VERO;
    }
  } else if (compara(val, a->val) <= 0) {
    // Caso ricorsivo: scendo a destra
    res = inserisci(a->dx, val);
  } else {
    // Caso ricorsivo: scendo a sinistra
    res = inserisci(a->sx, val);
  }
  return res;
}

boolean cerca(const Albero &a, coppia val) {
  boolean res = FALSO;
  if (vuoto(a) == FALSO) {
    int c = compara(val, a -> val);
    if (c == 0) {
      // Trovato
```

```
      res = VERO;
    } else if (c < 0) {
      // Scendo a destra
      res = cerca(a->dx, val);
    } else {
      // Scendo a sinistra
      res = cerca(a->sx, val);
    }
  }
  return res;
}

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

4 Scrivere all'interno del file `esercizio4.cc` la definizione della procedura `ordina` che, presi come parametri in ingresso un'array di interi `v` ed un intero `n`, corrispondente al numero di elementi ivi contenuti, ordini il contenuto dell'array stesso in modo `crescente`, implementando l'algoritmo di ordinamento `QUICKSORT`.

NOTA 1: La funzione `ordina` deve essere ricorsiva: una funzione è ricorsiva se invoca se stessa oppure se invoca altre funzioni ricorsive.

NOTA 2: Non è ammesso l'utilizzo di cicli iterativi nell'implementazione di `ordina`.

NOTA 3: È ammessa la dichiarazione e la definizione di funzioni ausiliarie.

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

2 **esercizio4.cc**

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

using namespace std;

const int DIM = 16;

void printarray(int v[], int min, int max);
void swap(int & a, int & b);
void sposta(int v[], int p, int u, int & piv);
void ordina1(int v[], int primo, int ultimo);
void ordina(int v[], int n);

int main() {
  // Esempio
  int myarray[DIM] = {41,3,9,1,5,17,6,20,37,2,8,23,10,0,11,19};

  printarray(myarray, 0, DIM - 1);
  cout << endl;
  ordina(myarray, DIM);
  printarray(myarray, 0, DIM - 1);
}

void printarray(int v[], int min, int max) {
  int i;
  cout << "[";
  for (i= min; i <= max;i++) {
    cout << setw(2) << v[i] << " ";
  }
  cout << "]\n";
}

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

void sposta(int v[], int p, int u, int & piv) {
  if (p >= u) {
    swap(v[p], v[piv]);
    piv=p;
  } else if (v[p] <= v[piv]) {
    sposta(v, p + 1, u, piv);
  } else if (v[u] >= v[piv]) {
    sposta(v, p, u - 1, piv);
  } else { // v[p]>v[piv]>v[u]
    swap(v[p], v[u]);
    sposta(v, p + 1, u - 1, piv);
  }
}
```

```
void ordina1(int v[], int primo, int ultimo) {
  if (primo < ultimo) {
    int piv = ultimo;
    sposta(v, primo, ultimo, piv);
    ordina1(v, primo, piv-1);
    ordina1(v, piv + 1, ultimo);
  }
}

void ordina(int v[],int n) {
  ordina1(v, 0, n - 1);
}
```