# Secondo Appello di Programmazione I

15 febbraio 2017
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 legga da un file di testo una sequenza di parole fino a che una di esse non sia la parola "STOP", o finché il file non termini in caso la parola non sia presente; le parole così lette vanno 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 "STOP" **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.

Ad esempio, se il file di input contenesse il seguente testo:

```
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 STOP 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.
```

allora il file in output conterrà:

```
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
```

NOTA 1: Il file di input va scorso dalla prima all'ultima parola, in sequenza; è ammesso (ma non richiesto obligatoriamente) leggere il file di input al massimo due volte.

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_STOP = "STOP";

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 ciclo di lettura, per stimare la dimensione del file
  while(!file_input.eof() && (strcmp(parola, PAROLA_STOP) != 0)) {
    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];
  // Secondo 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_STOP = "STOP";

void leggi_e_stampa(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) {
  // MP: 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 stop
    if(strcmp(parola, PAROLA_STOP) != 0) {
      // Chiamata ricorsiva
      leggi_e_stampa(in, out);
```

```
      // Salvo la parola sul file di output
      out << parola << " ";
    }
  }
}
```

1 Si legga da un file di testo una sequenza di parole fino a che una di esse non sia la parola "FINE", o finché il file non termini in caso la parola non sia presente; le parole così lette vanno 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_output> <file_input>
```

generi il contenuto del file `file_output` come descritto sopra. La parola chiave "FINE" **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.

Ad esempio, se il file di input contenesse il seguente testo:

```
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 FINE 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.
```

allora il file in output conterrà:

```
nisi laboris ullamco 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
```

NOTA 1: Il file di input va scorso dalla prima all'ultima parola, in sequenza; è ammesso (ma non richiesto obligatoriamente) leggere il file di input al massimo due volte.

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_STOP = "FINE";

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_output> <file_di_input>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[2], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[2] << ")\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 ciclo di lettura, per stimare la dimensione del file
  while(!file_input.eof() && (strcmp(parola, PAROLA_STOP) != 0)) {
    numero_parole++;
    file_input >> parola;
  }

  // Chiudo e riapro il file
  file_input.close();
  file_input.open(argv[2], ios::in);

  // Alloco lo spazio per salvare le parole in memoria
  char** parole = new char* [numero_parole];
  // Secondo 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[1], 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_STOP = "FINE";

void leggi_e_stampa(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_output> <file_di_input>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[2], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[2] << ")\n";
      exit(EXIT_FAILURE);
  }

  // Apertura file di output
  file_output.open(argv[1], 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 stop
    if(strcmp(parola, PAROLA_STOP) != 0) {
      // Chiamata ricorsiva
      leggi_e_stampa(in, out);
```

```
      // Salvo la parola sul file di output
      out << parola << " ";
    }
  }
}
```

1 Si legga da un file di testo una sequenza di parole fino a che una di esse non sia la parola "HALT", o finché il file non termini in caso la parola non sia presente; le parole così lette vanno 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 "HALT" **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.

Ad esempio, se il file di input contenesse il seguente testo:

```
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 HALT exercitation 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.
```

allora il file in output conterrà:

```
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
```

NOTA 1: Il file di input va scorso dalla prima all'ultima parola, in sequenza; è ammesso (ma non richiesto obbligatoriamente) leggere il file di input al massimo due volte.

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_STOP = "HALT";

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 ciclo di lettura, per stimare la dimensione del file
  while(!file_input.eof() && (strcmp(parola, PAROLA_STOP) != 0)) {
    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];
  // Secondo 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_STOP = "HALT";

void leggi_e_stampa(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) {
  // MP: 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 stop
    if(strcmp(parola, PAROLA_STOP) != 0) {
      // Chiamata ricorsiva
      leggi_e_stampa(in, out);
```

```
      // Salvo la parola sul file di output
      out << parola << " ";
    }
  }
}
```

1 Si legga da un file di testo una sequenza di parole fino a che una di esse non sia la parola "END", o finché il file non termini in caso la parola non sia presente; le parole così lette vanno 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_output> <file_input>
```

generi il contenuto del file `file_output` come descritto sopra. La parola chiave "END" **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.

Ad esempio, se il file di input contenesse il seguente testo:

```
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 END 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.
```

allora il file in output conterrà:

```
aliquip ut nisi laboris ullamco 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
```

NOTA 1: Il file di input va scorso dalla prima all'ultima parola, in sequenza; è ammesso (ma non richiesto obligatoriamente) leggere il file di input al massimo due volte.

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_STOP = "END";

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_output> <file_di_input>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[2], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[2] << ")\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 ciclo di lettura, per stimare la dimensione del file
  while(!file_input.eof() && (strcmp(parola, PAROLA_STOP) != 0)) {
    numero_parole++;
    file_input >> parola;
  }

  // Chiudo e riapro il file
  file_input.close();
  file_input.open(argv[2], ios::in);

  // Alloco lo spazio per salvare le parole in memoria
  char** parole = new char* [numero_parole];
  // Secondo 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[1], 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_STOP = "END";

void leggi_e_stampa(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_output> <file_di_input>" << endl;
      exit(EXIT_FAILURE);
  }

  // Tentativo di apertura file di input
  file_input.open(argv[2], ios::in);
  if (file_input.fail()) {
      cerr << "Non riesco ad accedere al file di input (" << argv[2] << ")\n";
      exit(EXIT_FAILURE);
  }

  // Apertura file di output
  file_output.open(argv[1], 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 stop
    if(strcmp(parola, PAROLA_STOP) != 0) {
      // Chiamata ricorsiva
      leggi_e_stampa(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` quadrati perfetti, cominciando dallo zero.

Per esempio, i primi 6 quadrati perfetti sono $0, 1, 4, 9, 16, 25$ (cioè rispettivamente $0^2, 1^2, 2^2, 3^2, 4^2, 5^2$).

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: 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 quadrati_perfetti(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* vector = crea_vettore(N);

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

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

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

void quadrati_perfetti(int v[], int n, int i) {
   if (i>= n) {
      return;
   } else {
      v[i] = i*i;
      quadrati_perfetti(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 triangolari, partendo da zero.

L'$i$-esimo numero triangolare si ottiene con la formula di Gauss

$$T(i) = \frac{i(i+1)}{2}.$$

Per esempio, i primi 8 numeri triangolari sono $0, 1, 3, 6, 10, 15, 21, 28$.

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: 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 numeri_triangolari(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* vector = crea_vettore(N);

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

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

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

void numeri_triangolari(int v[], int n, int i) {
   if (i>= n) {
      return;
   } else {
      v[i] = (i*(i+1))/2;
      numeri_triangolari(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 esagonali, partendo da zero.

L'$i$-esimo numero esagonale si ottiene con la formula $E(i) = i(2i - 1)$.

Per esempio, i primi 7 numeri esagonali sono $0, 1, 6, 15, 28, 45, 66$.

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: 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 numeri_esagonali(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* vector = crea_vettore(N);

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

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

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

void numeri_esagonali(int v[], int n, int i) {
   if (i>= n) {
      return;
   } else {
      v[i] = i*(2*i-1);
      numeri_esagonali(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 pentagonali, partendo da zero.

L'$i$-esimo numero pentagonale si ottiene con la seguente formula

$$P(i) = \frac{i(3i-1)}{2}.$$

Per esempio, i primi 9 numeri pentagonali sono $0, 1, 5, 12, 22, 35, 51, 70, 92$.

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: 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 numeri_pentagonali(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* vector = crea_vettore(N);

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

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

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

void numeri_pentagonali(int v[], int n, int i) {
   if (i>= n) {
      return;
   } else {
      v[i] = (i*(3*i-1))/2;
      numeri_pentagonali(v, n, i+1);
   }
}
```

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

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

La coda deve essere implementata con un array allocato dinamicamente, e il numero massimo di elementi che possono essere inseriti nella coda e' specificato dall'argomento `maxnum` della funzione `init`.

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

## 3 queue_main.cc

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

int main()
{
    char res;
    float num;
    queue q;
    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire il numero massimo di elementi della coda di float: ";
        cin >> maxdim;
    }

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

```
    } while (res != 'q');
    deinit(q);

    return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_QUEUE_H
#define STRUCT_QUEUE_H

struct queue {
    int head, tail;
    int dim;
    float *elem;
};


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

#endif
```

3 **queue.cc**

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

static int next(int index, const queue &q)
{
    return (index + 1) % q.dim;
}

void init(queue &q, int maxdim)
{
    q.tail = q.head = 0;
    q.dim = maxdim+1;
    q.elem = new float[q.dim];
}

void deinit(queue &q)
{
    delete[] q.elem;
}

static bool is_empty(const queue &q)
{
    return q.tail == q.head;
}
```

```
static bool is_full(const queue &q)
{
    return next(q.tail, q) == q.head;
}

bool enqueue(queue &q, float n)
{
    bool res = false;
    if (!is_full(q)) {
      q.elem[q.tail] = n;
      q.tail = next(q.tail, q);
      res = true;
    }
    return res;
}

bool dequeue(queue &q)
{
    bool res = false;
    if (!is_empty(q)) {
      q.head = next(q.head, q);
      res = true;
    }
    return res;
}

bool first(queue &q, float &out)
{
    bool res = false;
    if (!is_empty(q)) {
      out = q.elem[q.head];
      res = true;
    }
    return res;
}

void print(const queue &q)
{
    for (int i = q.head; i != q.tail; i = next(i, q)) {
        cout << q.elem[i] << endl;
    }
    cout << endl;
}
```

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

- `init` inizializzi la coda per contenere al più un numero massimo di elementi `dim` passato come parametro;
- `deinit` liberi la memoria utilizzata dalla coda;
- `accoda` inserisca l'elemento passato come parametro nella coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `testa` legga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `estrai_testa` tolga l'elemento in testa alla coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `stampa` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente andando a capo ad ogni elemento.

La coda deve essere implementata con un array allocato dinamicamente, e il numero massimo di elementi che possono essere inseriti nella coda e' specificato dall'argomento `maxnum` della funzione `init`.

NOTA: la dimensione dell'array dev'essere di 1 più grande, cioè q.dim = maxnum+1.

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

3 **queue_main.cc**

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

int main()
{
    char res;
    int num;
    queue q;
    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire il numero massimo di elementi della coda di interi: ";
        cin >> maxdim;
    }

    init(q, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << "Accoda (e)\n"
             << "Estrai testa (d)\n"
             << "Testa (f)\n"
             << "Stampa (p)\n"
             << "Quit (q)\n";
        cin >> res;
        switch (res) {
        case 'e':
            cout << "Valore: ";
            cin >> num;
            if (!accoda(q, num)) {
                cout << "Coda piena\n";
            }
            break;
        case 'd':
            if (!estrai_testa(q)) {
                cout << "Coda vuota\n";
            }
            break;
        case 'f':
            if (!testa(q, num)) {
                cout << "Coda vuota!\n";
            } else {
                cout << "First = " << num << endl;
            }
            break;
        case 'p':
            stampa(q);
            break;
        case 'q':
            break;
        default:
            cout << "Valore errato!\n";
        }
```

```
    } while (res != 'q');
    deinit(q);

    return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_CODA_H
#define STRUCT_CODA_H

struct queue {
    int head, tail;
    int size;
    int *elem;
};


void init(queue &q, int maxdim);
void deinit(queue &q);
bool accoda(queue &q, int n);
bool testa(queue &q, int &out);
bool estrai_testa(queue &q);
void stampa(const queue &q);

#endif
```

3 **queue.cc**

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

static int next(int index, const queue &q)
{
    return (index + 1) % q.size;
}

void init(queue &q, int maxdim)
{
    q.tail = q.head = 0;
    q.size = maxdim+1;
    q.elem = new int[q.size];
}

void deinit(queue &q)
{
    delete[] q.elem;
}

static bool vuota(const queue &q)
{
    return q.tail == q.head;
}
```

```
static bool piena(const queue &q)
{
    return next(q.tail, q) == q.head;
}

bool accoda(queue &q, int n)
{
    bool ris = false;
    if (!piena(q)) {
      q.elem[q.tail] = n;
      q.tail = next(q.tail, q);
      ris = true;
    }
    return ris;
}

bool estrai_testa(queue &q)
{
    bool ris = false;
    if (!vuota(q)) {
      q.head = next(q.head, q);
      ris = true;
    }
    return ris;
}

bool testa(queue &q, int &out)
{
    bool ris = false;
    if (!vuota(q)) {
      out = q.elem[q.head];
      ris = true;
    }
    return ris;
}

void stampa(const queue &q)
{
    for (int i = q.head; i != q.tail; i = next(i, q)) {
        cout << q.elem[i] << endl;
    }
    cout << endl;
}
```

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

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

La coda deve essere implementata con un array allocato dinamicamente, e il numero massimo di elementi che possono essere inseriti nella coda e' specificato dall'argomento `maxnum` della funzione `init`.

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

3 **queue_main.cc**

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

int main()
{
    char res;
    double num;
    queue q;
    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire il numero massimo di elementi della coda di double: ";
        cin >> maxdim;
    }

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

```
    } while (res != 'q');
    deinit(q);

    return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_QUEUE_H
#define STRUCT_QUEUE_H

struct queue {
    int head, tail;
    int size;
    double *elem;
};


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

#endif
```

3 **queue.cc**

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

static int next(int index, const queue &q)
{
    return (index + 1) % q.size;
}

void init(queue &q, int maxdim)
{
    q.tail = q.head = 0;
    q.size = maxdim+1;
    q.elem = new double[q.size];
}

void deinit(queue &q)
{
    delete[] q.elem;
}

static bool is_empty(const queue &q)
{
    return q.tail == q.head;
}
```

```
static bool is_full(const queue &q)
{
    return next(q.tail, q) == q.head;
}

bool enqueue(queue &q, double n)
{
    bool res = false;
    if (!is_full(q)) {
      q.elem[q.tail] = n;
      q.tail = next(q.tail, q);
      res = true;
    }
    return res;
}

bool dequeue(queue &q)
{
    bool res = false;
    if (!is_empty(q)) {
      q.head = next(q.head, q);
      res = true;
    }
    return res;
}

bool first(queue &q, double &out)
{
    bool res = false;
    if (!is_empty(q)) {
      out = q.elem[q.head];
      res = true;
    }
    return res;
}

void print(const queue &q)
{
    for (int i = q.head; i != q.tail; i = next(i, q)) {
        cout << q.elem[i] << endl;
    }
    cout << endl;
}
```

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

- `init` inizializzi la coda per contenere al più un numero massimo di elementi `dim` passato come parametro;
- `deinit` liberi la memoria utilizzata dalla coda;
- `stampa` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente andando a capo ad ogni elemento;
- `testa` legga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `estrai_testa` tolga l'elemento in testa alla coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `accoda` inserisca l'elemento passato come parametro nella coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti.

La coda deve essere implementata con un array allocato dinamicamente, e il numero massimo di elementi che possono essere inseriti nella coda e' specificato dall'argomento `maxnum` della funzione `init`.

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

3 **queue_main.cc**

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

int main()
{
    char res;
    char num;
    queue q;
    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire il numero massimo di elementi della coda di char: ";
        cin >> maxdim;
    }

    init(q, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << "Accoda (e)\n"
             << "Estrai testa (d)\n"
             << "Testa (f)\n"
             << "Stampa (p)\n"
             << "Quit (q)\n";
        cin >> res;
        switch (res) {
        case 'e':
            cout << "Valore: ";
            cin >> num;
            if (!accoda(q, num)) {
                cout << "Coda piena\n";
            }
            break;
        case 'd':
            if (!estrai_testa(q)) {
                cout << "Coda vuota\n";
            }
            break;
        case 'f':
            if (!testa(q, num)) {
                cout << "Coda vuota!\n";
            } else {
                cout << "First = " << num << endl;
            }
            break;
        case 'p':
            stampa(q);
            break;
        case 'q':
            break;
        default:
            cout << "Valore errato!\n";
        }
```

```
    } while (res != 'q');
    deinit(q);

    return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_CODA_H
#define STRUCT_CODA_H

struct queue {
    int head, tail;
    int dim;
    char *elem;
};


void init(queue &q, int maxdim);
void deinit(queue &q);
bool accoda(queue &q, char n);
bool testa(queue &q, char &out);
bool estrai_testa(queue &q);
void stampa(const queue &q);

#endif
```

3 **queue.cc**

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

static int next(int index, const queue &q)
{
    return (index + 1) % q.dim;
}

void init(queue &q, int maxdim)
{
    q.tail = q.head = 0;
    q.dim = maxdim+1;
    q.elem = new char[q.dim];
}

void deinit(queue &q)
{
    delete[] q.elem;
}

static bool vuota(const queue &q)
{
    return q.tail == q.head;
}
```

```
static bool piena(const queue &q)
{
    return next(q.tail, q) == q.head;
}

bool accoda(queue &q, char n)
{
    bool ris = false;
    if (!piena(q)) {
      q.elem[q.tail] = n;
      q.tail = next(q.tail, q);
      ris = true;
    }
    return ris;
}

bool estrai_testa(queue &q)
{
    bool ris = false;
    if (!vuota(q)) {
      q.head = next(q.head, q);
      ris = true;
    }
    return ris;
}

bool testa(queue &q, char &out)
{
    bool ris = false;
    if (!vuota(q)) {
      out = q.elem[q.head];
      ris = true;
    }
    return ris;
}

void stampa(const queue &q)
{
    for (int i = q.head; i != q.tail; i = next(i, q)) {
        cout << q.elem[i] << endl;
    }
    cout << endl;
}
```

4 La Torre di Hanoi è un gioco che consiste in una serie di tre pioli $0, 1, 2$ in cui infilare $N$ dischi di larghezze strettamente crescenti $1, 2, 3, ..., N$, in modo tale che un disco non possa mai stare sopra un altro disco di dimensione inferiore. Inizialmente tutti i dischi si trovano nel piolo 0, come nella figura di sinistra.

Attraverso una serie di spostamenti di un singolo disco alla volta bisogna portare il sistema nella situazione finale in cui tutti i dischi si trovano nel piolo 2, come nella figura di destra.

Siano dati i file `hanoi.h` e `hanoi.o` che, utilizzando a loro volta uno stack di interi, implementa il TDA **hanoi**, con le seguenti funzioni:

- `init(h, n)`: inizializza la torre di hanoi $h$ con $n$ dischi sul piolo 0;
- `print()`: stampa lo stato della torre di hanoi $h$;
- `move(h, s, t)`: nella torre di hanoi $h$ sposta il disco sulla cima al piolo $s$ alla cima del piolo $t$, restituendo "OK" se l'operazione è stata possibile secondo le regole, "FAIL" altrimenti.

Realizzare una funzione `moveall(h)` che porta la torre di hanoi $h$ dalla configurazione iniziale alla configurazione finale, come sopra descritte.

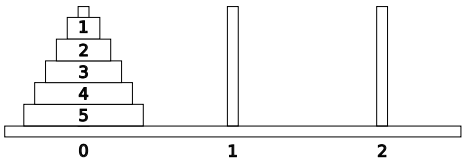


Figura 1: configurazione iniziale

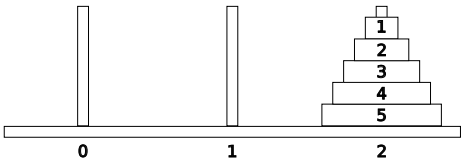


Figura 2: configurazione finale

2 **esercizio4.cc**

```
#include <iostream>
#include "struct_stack.h"
#include "hanoi.h"

retval moveall(hanoi &) ;

int main ()
{
  int ndiscs;
  hanoi h;
  cout << "numero di dischi? (< " << dim << "): ";
  cin >> ndiscs;
  init(h,ndiscs);
  cout << "\nInizialmente la disposizione e': " << endl;
  print(h);
  cout << endl;
  moveall(h);
  cout << "\nAlla fine la disposizione e': " << endl;
  print(h);
}

// INSERIRE QUI SOTTO LA FUNZIONE moveall
// ed eventuali funzioni ausiliarie
retval movemany(int n,hanoi & h, int source,int target,int intermediate) {

  retval res=OK;
  if (n!=0) {
    movemany(n-1,h,source,intermediate,target);
    move(h,source,target);
// cout << "Sposto il disco da posizione " << source
//      << " a posizione " << target << ":" << endl;
// print(h);
    movemany(n-1,h,intermediate,target,source);
  }
  return res;
}

retval moveall(hanoi & h) {
  return movemany(h.ndiscs,h,0,2,1);
}
```