Quinto Appello di Programmazione I

04 Settembre 2012
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 esame

Durante la prova gli studenti sono vincolati a seguire le regole seguenti:

- Non è consentito l'uso di alcun libro di testo o fotocopia. In caso lo studente necessitasse di carta (?), gli/le verranno forniti fogli di carta bianca su richiesta, che dovranno essere riconsegnati a fine prova. È consentito l'uso di una penna. Non è consentito l'uso di alcuno strumento calcolatore.
- È vietato lo scambio di qualsiasi informazione, orale o scritta. È vietato guardare nel terminale del vicino.
- È vietato l'uso di telefoni cellulari o di qualsiasi strumento elettronico.
- È vietato allontanarsi dall'aula durante la prova, anche se si ha già consegnato. (Ogni necessità fisiologica va espletata PRIMA dell'inizio della prova.)
- È vietato qualunque accesso, in lettura o scrittura, a file esterni alla directory di lavoro assegnata a ciascun studente. Le uniche operazioni consentite sono l'apertura, l'editing, la copia, la rimozione e la compilazione di file all'interno della propria directory di lavoro.
- Sono ovviamente vietati l'uso di email, ftp, ssh, telnet ed ogni strumento che consenta di accedere a file esterni alla directory di lavoro. Le operazioni di copia, rimozione e spostamento di file devono essere circoscritte alla directory di lavoro.
- Ogni altra attività non espressamente citata qui sopra o autorizzata dal docente è vietata.

Ogni violazione delle regole di cui sopra comporterà automaticamente l'annullamento della prova e il divieto di accesso ad un certo numero di appelli successivi, a seconda della gravità e della recidività della violazione.

NOTA IMPORTANTE: DURANTE LA PROVA PER OGNI STUDENTE VERRÀ ATTIVATO UN TRACCIATORE SOFTWARE CHE REGISTRERÀ TUTTE LE OPERAZIONI ESEGUITE (ANCHE ALL'INTERNO DELL'EDITOR!!). L'ANNULLAMENTO DELLA PROVA DI UNO STUDENTE POTRÀ AVVENIRE ANCHE IN UN SECONDO MOMENTO, SE L'ANALISI DELLE TRACCE SOFTWARE RIVELASSERO IRREGOLARITÀ.

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main`:

- il nome *file1* di un file di testo di input, contenente una o più righe
- il nome *file2* di un file di testo di output

legga il file di testo di nome *file1* (producendo il messaggio di errore "File inesistente\n" se non riesce ad aprirlo) e crei un nuovo file di testo di nome *file2* contenente la lista delle dimensioni di ciascuna riga del file *file1*. Si richiede inoltre di stampare a video la riga **più lunga** e la sua dimensione.

Ogni numero va stampato su di una riga distinta. Se ad esempio l'eseguibile è `a.out` ed il file `input.txt` ha il seguente contenuto:

```
Filastrocca delle parole:
Fatevi avanti! Chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
Buon giorno, Buon anno, Siate felici!
Parole belle e parole buone;
parole per ogni sorta di persone.
Di G. Rodari.
```

allora il comando:

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

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

```
25
28
...
13
```

E stampa a video:

```
Buon giorno, Buon anno, Siate felici!
37
```

NOTA 1: Per dimensione di una riga si intende il numero dei caratteri ivi contenuti, escluso il carattere "a capo". Spazi e segni di punteggiatura vanno conteggiati nel computo dei caratteri di ogni riga.

NOTA 2: Per semplicità si assuma che ogni riga contenga al più 255 caratteri.

NOTA 3: non è ammesso l'uso di alcuna routine di gestione di stringhe (esempio **strlen**, **strcpy**); è invece ammesso l'uso di routine di gestione di singoli caratteri (ad esempio `get` e `put`).

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

1 **esercizio_A11.cc**

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

using namespace std;

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

  fstream my_in, my_out;
  char buf[256], str[256];

  // Controllo di sintassi
  if(argc != 3) {
    cout << "Sintassi: ./a.out <file1> <file2>\n";
    exit(-1);
  }

  // Prova ad aprire il file di input
  my_in.open(argv[1],ios::in);
  // Verifica se l'apertura e' andata a buon fine
  if(my_in.fail()) {
    cout << "File non esistente\n";
    exit(-1);
  }
  // Apre il file di output
  my_out.open(argv[2],ios::out);

  // Legge una riga
  my_in.getline(buf, 256);
  // La dimensione massima e' impostata al valore piu' piccolo
  // rappresentabile
  int max_len = numeric_limits<int>::min();

  while(!my_in.eof()) {
    // Calcola lunghezza della stringa "buf"
    int len = 0;
    while(buf[len] != '\0') {
      len++;
    }
    // Salva la dimensione della riga nel file di output
    my_out << len << endl;
    if(len > max_len) {
      // Memorizza la lunghezza massima
      max_len = len;
      // Salva la stringa come stringa piu' lunga
      for(int i = 0; i < len; i++) {
        str[i] = buf[i];
      }
      // Terminatore stringa
      str[len] = '\0';
    }
```

```
    // Legge una riga
    my_in.getline(buf, 256);
  }

  // Chiude i file
  my_out.close();
  my_in.close();

  // Stampa a video il risultato
  cout << str << endl;
  cout << max_len << endl;

  return(0);
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main`:

- il nome *file1* di un file di testo di input, contenente una o più righe
- il nome *file2* di un file di testo di output

legga il file di testo di nome *file1* (producendo il messaggio di errore "File inesistente\n" se non riesce ad aprirlo) e crei un nuovo file di testo di nome *file2* contenente la lista delle dimensioni di ciascuna riga del file *file1*. Si richiede inoltre di stampare a video la riga **più corta** e la sua dimensione.

Ogni numero va stampato su di una riga distinta. Se ad esempio l'eseguibile è `a.out` ed il file `input.txt` ha il seguente contenuto:

```
Filastrocca delle parole:
Fatevi avanti! Chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
Buon giorno, Buon anno, Siate felici!
Parole belle e parole buone;
parole per ogni sorta di persone.
Di G. Rodari.
```

allora il comando:

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

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

```
25
28
...
13
```

E stampa a video:

```
Di G. Rodari.
13
```

NOTA 1: Per dimensione di una riga si intende il numero dei caratteri ivi contenuti, escluso il carattere "a capo". Spazi e segni di punteggiatura vanno conteggiati nel computo dei caratteri di ogni riga.

NOTA 2: Per semplicità si assuma che ogni riga contenga al più 255 caratteri.

NOTA 3: non è ammesso l'uso di alcuna routine di gestione di stringhe (esempio **strlen**, **strcpy**); è invece ammesso l'uso di routine di gestione di singoli caratteri (ad esempio `get` e `put`).

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

1 **esercizio_A12.cc**

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

using namespace std;

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

  fstream my_in, my_out;
  char buf[256], str[256];

  // Controllo di sintassi
  if(argc != 3) {
    cout << "Sintassi: ./a.out <file1> <file2>\n";
    exit(-1);
  }

  // Prova ad aprire il file di input
  my_in.open(argv[1],ios::in);
  // Verifica se l'apertura e' andata a buon fine
  if(my_in.fail()) {
    cout << "File non esistente\n";
    exit(-1);
  }
  // Apre il file di output
  my_out.open(argv[2],ios::out);

  // Legge una riga
  my_in.getline(buf, 256);
  // La dimensione minima e' impostata al valore piu' grande
  // rappresentabile
  int min_len = numeric_limits<int>::max();

  while(!my_in.eof()) {
    // Calcola lunghezza della stringa "buf"
    int len = 0;
    while(buf[len] != '\0') {
      len++;
    }
    // Salva la dimensione della riga nel file di output
    my_out << len << endl;
    if(len < min_len) {
      // Memorizza la lunghezza minima
      min_len = len;
      // Salva la stringa come stringa piu' corta
      for(int i = 0; i < len; i++) {
        str[i] = buf[i];
      }
      // Terminatore stringa
      str[len] = '\0';
    }
```

```
    // Legge una riga
    my_in.getline(buf, 256);
  }

  // Chiude i file
  my_out.close();
  my_in.close();

  // Stampa a video il risultato
  cout << str << endl;
  cout << min_len << endl;

  return(0);
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main`:

- il nome *file1* di un file di testo di input, contenente una o più righe
- il nome *file2* di un file di testo di output

legga il file di testo di nome *file1* (producendo il messaggio di errore "File inesistente\n" se non riesce ad aprirlo) e crei un nuovo file di testo di nome *file2* contenente la lista delle dimensioni di ciascuna riga del file *file1*. Si richiede inoltre di stampare a video la **prima** riga e la dimensione di quella **più lunga**.

Ogni numero va stampato su di una riga distinta. Se ad esempio l'eseguibile è `a.out` ed il file `input.txt` ha il seguente contenuto:

```
Filastrocca delle parole:
Fatevi avanti! Chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
Buon giorno, Buon anno, Siate felici!
Parole belle e parole buone;
parole per ogni sorta di persone.
Di G. Rodari.
```

allora il comando:

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

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

```
25
28
...
13
```

E stampa a video:

```
Filastrocca delle parole:
37
```

NOTA 1: Per dimensione di una riga si intende il numero dei caratteri ivi contenuti, escluso il carattere "a capo". Spazi e segni di punteggiatura vanno conteggiati nel computo dei caratteri di ogni riga.

NOTA 2: Per semplicità si assuma che ogni riga contenga al più 255 caratteri.

NOTA 3: non è ammesso l'uso di alcuna routine di gestione di stringhe (esempio **strlen**, **strcpy**); è invece ammesso l'uso di routine di gestione di singoli caratteri (ad esempio `get` e `put`).

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

1 **esercizio_A13.cc**

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

using namespace std;

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

  fstream my_in, my_out;
  char buf[256], str[256];

  // Controllo di sintassi
  if(argc != 3) {
    cout << "Sintassi: ./a.out <file1> <file2>\n";
    exit(-1);
  }

  // Prova ad aprire il file di input
  my_in.open(argv[1],ios::in);
  // Verifica se l'apertura e' andata a buon fine
  if(my_in.fail()) {
    cout << "File non esistente\n";
    exit(-1);
  }
  // Apre il file di output
  my_out.open(argv[2],ios::out);

  // Legge una riga
  my_in.getline(buf, 256);
  // La dimensione massima e' impostata al valore piu' piccolo
  // rappresentabile
  int max_len = numeric_limits<int>::min();

  // Flag per indicare se abbiamo gia' salvato la prima riga
  bool prima = false;
  while(!my_in.eof()) {
    // Calcola lunghezza della stringa "buf"
    int len = 0;
    while(buf[len] != '\0') {
      len++;
    }
    // Salva la stringa come prima riga se necessario
    if(!prima) {
      for(int i = 0; i < len; i++) {
        str[i] = buf[i];
      }
      // Terminatore stringa
      str[len] = '\0';
      // Setta il flag
      prima = true;
    }
```

```
    // Salva la dimensione della riga nel file di output
    my_out << len << endl;
    if(len > max_len) {
      // Memorizza la lunghezza massima
      max_len = len;
    }
    // Legge una riga
    my_in.getline(buf, 256);
  }

  // Chiude i file
  my_out.close();
  my_in.close();

  // Stampa a video il risultato
  cout << str << endl;
  cout << max_len << endl;

  return(0);
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main`:

- il nome *file1* di un file di testo di input, contenente una o più righe
- il nome *file2* di un file di testo di output

legga il file di testo di nome *file1* (producendo il messaggio di errore "File inesistente\n" se non riesce ad aprirlo) e crei un nuovo file di testo di nome *file2* contenente la lista delle dimensioni di ciascuna riga del file *file1*. Si richiede inoltre di stampare a video la riga **prima** riga e la dimensione di quella **più corta**.

Ogni numero va stampato su di una riga distinta. Se ad esempio l'eseguibile è `a.out` ed il file `input.txt` ha il seguente contenuto:

```
Filastrocca delle parole:
Fatevi avanti! Chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
Buon giorno, Buon anno, Siate felici!
Parole belle e parole buone;
parole per ogni sorta di persone.
Di G. Rodari.
```

allora il comando:

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

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

```
25
28
...
13
```

E stampa a video:

```
Filastrocca delle parole:
13
```

NOTA 1: Per dimensione di una riga si intende il numero dei caratteri ivi contenuti, escluso il carattere "a capo". Spazi e segni di punteggiatura vanno conteggiati nel computo dei caratteri di ogni riga.

NOTA 2: Per semplicità si assuma che ogni riga contenga al più 255 caratteri.

NOTA 3: non è ammesso l'uso di alcuna routine di gestione di stringhe (esempio **strlen**, **strcpy**); è invece ammesso l'uso di routine di gestione di singoli caratteri (ad esempio `get` e `put`).

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

1 **esercizio_A14.cc**

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

using namespace std;

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

  fstream my_in, my_out;
  char buf[256], str[256];

  // Controllo di sintassi
  if(argc != 3) {
    cout << "Sintassi: ./a.out <file1> <file2>\n";
    exit(-1);
  }

  // Prova ad aprire il file di input
  my_in.open(argv[1],ios::in);
  // Verifica se l'apertura e' andata a buon fine
  if(my_in.fail()) {
    cout << "File non esistente\n";
    exit(-1);
  }
  // Apre il file di output
  my_out.open(argv[2],ios::out);

  // Legge una riga
  my_in.getline(buf, 256);
  // La dimensione minima e' impostata al valore piu' grande
  // rappresentabile
  int min_len = numeric_limits<int>::max();

  // Flag per indicare se abbiamo gia' salvato la prima riga
  bool prima = false;
  while(!my_in.eof()) {
    // Calcola lunghezza della stringa "buf"
    int len = 0;
    while(buf[len] != '\0') {
      len++;
    }
    // Salva la stringa come prima riga se necessario
    if(!prima) {
      for(int i = 0; i < len; i++) {
        str[i] = buf[i];
      }
      // Terminatore stringa
      str[len] = '\0';
      // Setta il flag
      prima = true;
    }
```

```
    // Salva la dimensione della riga nel file di output
    my_out << len << endl;
    if(len < min_len) {
      // Memorizza la lunghezza minima
      min_len = len;
    }
    // Legge una riga
    my_in.getline(buf, 256);
  }

  // Chiude i file
  my_out.close();
  my_in.close();

  // Stampa a video il risultato
  cout << str << endl;
  cout << min_len << endl;

  return(0);
}
```

1 Nel file `esercizio2.cc` sono contenute le seguenti funzioni:

- `insertion_sort_ric` e `get_index_of_min_ric` che implementano in modo ricorsivo l'algoritmo di ordinamento *insertion sort*;
- `main` che (a) inizializza un array `V` di **N** interi, (b) stampa a video il contenuto di `V`, (c) invoca la funzione `insertion_sort_it` per ordinare `V` in modo crescente ed infine (d) stampa il contenuto di `V` per mostrare il risultato dell'ordinamento.

Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione `insertion_sort_it` che deve implementare *in modo iterativo* l'algoritmo *insertion sort* definito dalle funzioni `insertion_sort_ric` e `get_index_of_min_ric`.

NOTA 1: L'esercizio richiede di riscrivere la funzione `insertion_sort_ric` in modo iterativo.

NOTA 2: La funzione `insertion_sort_it` non può essere ricorsiva: al suo interno non ci possono quindi essere chiamate a sè stessa o ad altre funzioni ricorsive.

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

2 **esercizio_A21.cc**

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

const int N = 15;

// Inserire qui sotto la dichiarazione della funzione insertion_sort_it
void insertion_sort_it(int v[], int dim);

int get_index_of_min_ric(int v[], int dim, int pos, int pos_min) {
   if (pos >= dim) {
      return pos_min;
   } else if (v[pos] < v[pos_min]) {
      return get_index_of_min_ric(v, dim, pos+1, pos);
   } else {
      return get_index_of_min_ric(v, dim, pos+1, pos_min);
   }
}

// Esempio di invocazione: insertion_sort_ric(vector,N, 0);
void insertion_sort_ric(int v[], int d, int p) {
  if (p >= d-1) {
    return;
  } else {
    int pos_min = get_index_of_min_ric(v,d,p,p);
    int tmp = v[p];
    v[p] = v[pos_min];
    v[pos_min] = tmp;
    insertion_sort_ric (v, d, p+1);
  }
}

int main(){

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

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

   insertion_sort_it(vector,N);

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

return 0;
}
```

```
// Inserire qui sotto la definizione della funzione insertion_sort_it

void insertion_sort_it(int v[], int dim) {
   int min, i;
   for (i=0; i<dim-1; i++) {
      min=i;
      for (int j=i; j<dim; j++) {
         if (v[j] < v[min]) {
            min = j;
         }
      }
      int tmp = v[i];
      v[i] = v[min];
      v[min] = tmp;
   }
}
```

2 Nel file `esercizio2.cc` sono contenute le seguenti funzioni:

- `insertion_sort_ric` e `get_index_of_max_ric` che implementano in modo ricorsivo l'algoritmo di ordinamento *insertion sort*;
- `main` che (a) inizializza un array `V` di **N** *double*, (b) stampa a video il contenuto di `V`, (c) invoca la funzione `insertion_sort_it` per ordinare `V` in modo decrescente ed infine (d) stampa il contenuto di `V` per mostrare il risultato dell'ordinamento.

Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione `insertion_sort_it` che deve implementare *in modo iterativo* l'algoritmo *insertion sort* definito dalle funzioni `insertion_sort_ric` e `get_index_of_max_ric`.

NOTA 1: L'esercizio richiede di riscrivere la funzione `insertion_sort_ric` in modo iterativo.

NOTA 2: La funzione `insertion_sort_it` non può essere ricorsiva: al suo interno non ci possono quindi essere chiamate a sè stessa o ad altre funzioni ricorsive.

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

1 **esercizio_A22.cc**

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

const int N = 15;

// Inserire qui sotto la dichiarazione della funzione insertion_sort_it
void insertion_sort_it(double v[], int dim);

int get_index_of_max_ric(double v[], int dim, int pos, int pos_max) {
   if (pos >= dim) {
      return pos_max;
   } else if (v[pos] > v[pos_max]) {
      return get_index_of_max_ric(v, dim, pos+1, pos_max);
   } else {
      return get_index_of_max_ric(v, dim, pos+1, pos);
   }
}

// Esempio di invocazione: insertion_sort_ric(vector,N, 0);
void insertion_sort_ric(double v[], int d, int p) {
  if (p >= d-1) {
    return;
  } else {
    int pos_max = get_index_of_max_ric(v,d,p,p);
    double tmp = v[p];
    v[p] = v[pos_max];
    v[pos_max] = tmp;
    insertion_sort_ric (v, d, p+1);
  }
}

int main(){

   double vector[] = {17.2, 41.1, 202.3, 5.0, 13.7, 25.4, 2.1, 7.5,
                      8.1, 161.9, 59.9, 73.2, 88.0, 76.2, 13.5};

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

   insertion_sort_it(vector,N);

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

return 0;
}
```

```
// Inserire qui sotto la definizione della funzione insertion_sort_it

void insertion_sort_it(double v[], int dim) {
   int max, i;
   for (i=0; i<dim-1; i++) {
      max=i;
      for (int j=i; j<dim; j++) {
         if (v[j] > v[max]) {
            max = j;
         }
      }
      double tmp = v[i];
      v[i] = v[max];
      v[max] = tmp;
   }
}
```

1 Nel file `esercizio2.cc` sono contenute le seguenti funzioni:

- `insertion_sort_ric` e `get_index_of_min_ric` che implementano in modo ricorsivo l'algoritmo di ordinamento *insertion sort*;
- `main` che (a) inizializza un array `V` di **N** *float*, (b) stampa a video il contenuto di `V`, (c) invoca la funzione `insertion_sort_it` per ordinare `V` in modo crescente ed infine (d) stampa il contenuto di `V` per mostrare il risultato dell'ordinamento.

Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione `insertion_sort_it` che deve implementare *in modo iterativo* l'algoritmo *insertion sort* definito dalle funzioni `insertion_sort_ric` e `get_index_of_min_ric`.

NOTA 1: L'esercizio richiede di riscrivere la funzione `insertion_sort_ric` in modo iterativo.

NOTA 2: La funzione `insertion_sort_it` non può essere ricorsiva: al suo interno non ci possono quindi essere chiamate a sè stessa o ad altre funzioni ricorsive.

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

1 **esercizio_A23.cc**

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

const int N = 15;

// Inserire qui sotto la dichiarazione della funzione insertion_sort_it
void insertion_sort_it(float v[], int dim);

int get_index_of_min_ric(float v[], int dim, int pos, int pos_min) {
   if (pos >= dim) {
      return pos_min;
   } else if (v[pos] < v[pos_min]) {
      return get_index_of_min_ric(v, dim, pos+1, pos);
   } else {
      return get_index_of_min_ric(v, dim, pos+1, pos_min);
   }
}

// Esempio di invocazione: insertion_sort_ric(vector,N, 0);
void insertion_sort_ric(float v[], int d, int p) {
  if (p >= d-1) {
    return;
  } else {
    int pos_min = get_index_of_min_ric(v,d,p,p);
    float tmp = v[p];
    v[p] = v[pos_min];
    v[pos_min] = tmp;
    insertion_sort_ric (v, d, p+1);
  }
}

int main(){

   float vector[] = {22.35, 72.8, 41.7, 2.0, 51.3, 31.9, 22.2,
                     7.1, 8.9, 101.0, 52.5, 88.1, 77.6, 92.2, 20.2};

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

   insertion_sort_it(vector,N);

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

return 0;
}
```

```
// Inserire qui sotto la definizione della funzione insertion_sort_it

void insertion_sort_it(float v[], int dim) {
   int min, i;
   for (i=0; i<dim-1; i++) {
      min=i;
      for (int j=i; j<dim; j++) {
         if (v[j] < v[min]) {
            min = j;
         }
      }
      float tmp = v[i];
      v[i] = v[min];
      v[min] = tmp;
   }
}
```

1 Nel file `esercizio2.cc` sono contenute le seguenti funzioni:

- `insertion_sort_ric` e `get_index_of_max_ric` che implementano in modo ricorsivo l'algoritmo di ordinamento *insertion sort*;
- `main` che (a) inizializza un array `V` di **N** interi *long*, (b) stampa a video il contenuto di `V`, (c) invoca la funzione `insertion_sort_it` per ordinare `V` in modo decrescente ed infine (d) stampa il contenuto di `V` per mostrare il risultato dell'ordinamento.

Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione `insertion_sort_it` che deve implementare *in modo iterativo* l'algoritmo *insertion sort* definito dalle funzioni `insertion_sort_ric` e `get_index_of_max_ric`.

NOTA 1: L'esercizio richiede di riscrivere la funzione `insertion_sort_ric` in modo iterativo.

NOTA 2: La funzione `insertion_sort_it` non può essere ricorsiva: al suo interno non ci possono quindi essere chiamate a sè stessa o ad altre funzioni ricorsive.

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

1 **esercizio_A24.cc**

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

const int N = 15;

// Inserire qui sotto la dichiarazione della funzione insertion_sort_it
void insertion_sort_it(long v[], int dim);

int get_index_of_max_ric(long v[], int dim, int pos, int pos_max) {
   if (pos >= dim) {
      return pos_max;
   } else if (v[pos] > v[pos_max]) {
      return get_index_of_max_ric(v, dim, pos+1, pos_max);
   } else {
      return get_index_of_max_ric(v, dim, pos+1, pos);
   }
}

// Esempio di invocazione: insertion_sort_ric(vector,N, 0);
void insertion_sort_ric(long v[], int d, int p) {
  if (p >= d-1) {
    return;
  } else {
    int pos_max = get_index_of_max_ric(v,d,p,p);
    long tmp = v[p];
    v[p] = v[pos_max];
    v[pos_max] = tmp;
    insertion_sort_ric (v, d, p+1);
  }
}

int main(){

   long vector[] = {13, 22, 51, 113, 16, 89, 1, 170, 6, 12, 42, 72, 18, 10, 33};

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

   insertion_sort_it(vector,N);

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

return 0;
}
```

```
// Inserire qui sotto la definizione della funzione insertion_sort_it

void insertion_sort_it(long v[], int dim) {
   int max, i;
   for (i=0; i<dim-1; i++) {
      max=i;
      for (int j=i; j<dim; j++) {
         if (v[j] > v[max]) {
            max = j;
         }
      }
      long tmp = v[i];
      v[i] = v[max];
      v[max] = tmp;
   }
}
```

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

- `init` inizializzi la coda;
- `empty` restituisca `FALSE` se la coda contiene almeno un valore, e `TRUE` altrimenti;
- `enqueue` inserisca il valore passato come parametro nella coda;
- `dequeue` elimini il valore in testa alla coda, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `first` legga il valore 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 (senza modificarla), nell'ordine in cui i valori contenuti ne verrebbero estratti.

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

NOTA: ad eccezione della operazione `print`, si ricorda che ogni altra operazione sulla coda deve avere costo costante, indipendentemente dal numero di elementi contenuti nella struttura dati.

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

3 queue_A31_main.cc

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

#include "queue.h"


int main ()
{
    char scelta;
    queue q;

    long val;

    init(q);

    do
    {
        cout << "Operazioni possibili:\n"
             << "e : enqueue\n"
             << "d : dequeue\n"
             << "f : first\n"
             << "p : print\n"
             << "u : uscita\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'e':
            cout << "Valore da inserire? ";
            cin >> val;
            enqueue(q, val);
            break;
          case 'd':
            if (! dequeue(q))
                cout << "Coda vuota!\n";
            else
                cout << "Dequeue ok!\n";
            break;
          case 'f':
            if (! first(q, val))
                cout << "Coda vuota!\n";
            else
              cout << "Primo elemento della coda: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto della coda: ";
            print(q);
            break;
          default:
            if (scelta != 'u')
                cout << "Operazione non valida: " << scelta << endl;
        }
    } while (scelta != 'u');
```

```
    return 0;
}
```

3 **queue_A31.h**

```
#ifndef QUEUE_H
 #define QUEUE_H


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

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


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


void   init    (queue &q);
void   enqueue (queue &q, long val);
retval dequeue (queue &q);
retval empty   (const queue &q);
retval first   (const queue &q, long &result);
void   print   (const queue &q);


#endif // QUEUE_H
```

3 **queue_A31.cc**

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

#include "queue.h"


void init (queue &q)
{
    q.head = q.tail = NULL;
}

retval empty (const queue &q)
{
    return (q.head == NULL ? TRUE : FALSE);
}

void enqueue (queue &q, long val)
```

```
{
    node *n = new node;

    n->val = val;
    n->next = NULL;
    if (empty(q))
      q.head = q.tail = n;
    else
    {
      q.tail->next = n;
      q.tail = n;
    }
}

retval dequeue (queue &q)
{
    node *first;
    if (empty(q))
      return FALSE;

    first = q.head;
    q.head = q.head->next;
    delete first;

    if (empty(q))
      q.tail = NULL;

    return TRUE;
}

retval first (const queue &q, long &result)
{
    if (empty(q))
      return FALSE;

    result = q.head->val;
    return TRUE;
}


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

3 Nel file `queue_main.cc` è definita la funzione `main` che contiene un menu per gestire una coda di caratteri `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;
- `empty` restituisca `FALSE` se la coda contiene almeno un carattere, e `TRUE` altrimenti;
- `enqueue` inserisca il carattere passato come parametro nella coda;
- `dequeue` elimini il carattere in testa alla coda, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `first` legga il carattere 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 (senza modificarla), nell'ordine in cui i valori contenuti ne verrebbero estratti.

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

NOTA: ad eccezione della operazione `print`, si ricorda che ogni altra operazione sulla coda deve avere costo costante, indipendentemente dal numero di elementi contenuti nella struttura dati.

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

3 queue_A32_main.cc

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

#include "queue.h"


int main ()
{
    char scelta;
    queue q;

    char val;

    init(q);

    do
    {
        cout << "Operazioni possibili:\n"
             << "e : enqueue\n"
             << "d : dequeue\n"
             << "f : first\n"
             << "p : print\n"
             << "u : uscita\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'e':
            cout << "Valore da inserire? ";
            cin >> val;
            enqueue(q, val);
            break;
          case 'd':
            if (! dequeue(q))
                cout << "Coda vuota!\n";
            else
                cout << "Dequeue ok!\n";
            break;
          case 'f':
            if (! first(q, val))
                cout << "Coda vuota!\n";
            else
              cout << "Primo elemento della coda: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto della coda: ";
            print(q);
            break;
          default:
            if (scelta != 'u')
                cout << "Operazione non valida: " << scelta << endl;
        }
    } while (scelta != 'u');
```

```
    return 0;
}
```

3 **queue_A32.h**

```
#ifndef QUEUE_H
 #define QUEUE_H


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

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


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


void   init    (queue &q);
void   enqueue (queue &q, char val);
retval dequeue (queue &q);
retval empty   (const queue &q);
retval first   (const queue &q, char &result);
void   print   (const queue &q);


#endif // QUEUE_H
```

3 **queue_A32.cc**

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

#include "queue.h"


void init (queue &q)
{
    q.head = q.tail = NULL;
}

retval empty (const queue &q)
{
    return (q.head == NULL ? TRUE : FALSE);
}

void enqueue (queue &q, char val)
```

```
{
    node *n = new node;

    n->val = val;
    n->next = NULL;
    if (empty(q))
      q.head = q.tail = n;
    else
    {
      q.tail->next = n;
      q.tail = n;
    }
}

retval dequeue (queue &q)
{
    node *first;
    if (empty(q))
      return FALSE;

    first = q.head;
    q.head = q.head->next;
    delete first;

    if (empty(q))
      q.tail = NULL;

    return TRUE;
}

retval first (const queue &q, char &result)
{
    if (empty(q))
      return FALSE;

    result = q.head->val;
    return TRUE;
}


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

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

- `crea` inizializzi la coda;
- `vuota` restituisca `FALSE` se la coda contiene almeno un valore, e `TRUE` altrimenti;
- `inserisci` inserisca il valore passato come parametro nella coda;
- `elimina` elimini il valore in testa alla coda, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `primo` legga il valore 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;
- `stampa` stampi a video il contenuto della coda (senza modificarla), nell'ordine in cui i valori contenuti ne verrebbero estratti.

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

NOTA: ad eccezione della operazione `stampa`, si ricorda che ogni altra operazione sulla coda deve avere costo costante, indipendentemente dal numero di elementi contenuti nella struttura dati.

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

3 queue_A33_main.cc

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

#include "queue.h"


int main ()
{
    char scelta;
    queue q;

    double val;

    crea(q);

    do
    {
        cout << "Operazioni possibili:\n"
             << "i : inserisci\n"
             << "e : elimina\n"
             << "p : primo\n"
             << "s : stampa\n"
             << "u : uscita\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'i':
            cout << "Valore da inserire? ";
            cin >> val;
            inserisci(q, val);
            break;
          case 'e':
            if (! elimina(q))
                cout << "Coda vuota!\n";
            else
                cout << "Eliminazione ok!\n";
            break;
          case 'p':
            if (! primo(q, val))
                cout << "Coda vuota!\n";
            else
              cout << "Primo elemento della coda: " << val << endl;
            break;
          case 's':
            cout << "Contenuto della coda: ";
            stampa(q);
            break;
          default:
            if (scelta != 'u')
                cout << "Operazione non valida: " << scelta << endl;
        }
    } while (scelta != 'u');
```

```
    return 0;
}
```

3 **queue_A33.h**

```
#ifndef QUEUE_H
 #define QUEUE_H


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

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


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


void   crea (queue &q);
void   inserisci (queue &q, double val);
retval elimina (queue &q);
retval vuota   (const queue &q);
retval primo   (const queue &q, double &result);
void   stampa   (const queue &q);


#endif // QUEUE_H
```

2 **queue_A33.cc**

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

#include "queue.h"


void crea (queue &q)
{
    q.head = q.tail = NULL;
}

retval vuota (const queue &q)
{
    return (q.head == NULL ? TRUE : FALSE);
}

void inserisci (queue &q, double val)
```

```
{
    node *n = new node;

    n->val = val;
    n->next = NULL;
    if (vuota(q))
      q.head = q.tail = n;
    else
    {
      q.tail->next = n;
      q.tail = n;
    }
}

retval elimina (queue &q)
{
    node *first;
    if (vuota(q))
      return FALSE;

    first = q.head;
    q.head = q.head->next;
    delete first;

    if (vuota(q))
      q.tail = NULL;

    return TRUE;
}

retval primo (const queue &q, double &result)
{
    if (vuota(q))
      return FALSE;

    result = q.head->val;
    return TRUE;
}


void stampa (const queue &q)
{
    node *n = q.head;
    while (n != NULL)
    {
      cout << n->val << " ";
      n = n->next;
    }
    cout << endl;
}
```

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

- `crea` inizializzi la coda;
- `vuota` restituisca `FALSE` se la coda contiene almeno un numero, e `TRUE` altrimenti;
- `inserisci` inserisca il numero passato come parametro nella coda;
- `elimina` elimini il numero in testa alla coda, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `primo` legga il numero 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;
- `stampa` stampi a video il contenuto della coda (senza modificarla), nell'ordine in cui i valori contenuti ne verrebbero estratti.

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

NOTA: ad eccezione della operazione `stampa`, si ricorda che ogni altra operazione sulla coda deve avere costo costante, indipendentemente dal numero di elementi contenuti nella struttura dati.

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

3 queue_A34_main.cc

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

#include "queue.h"


int main ()
{
    char scelta;
    queue q;

    float val;

    crea(q);

    do
    {
        cout << "Operazioni possibili:\n"
             << "i : inserisci\n"
             << "e : elimina\n"
             << "p : primo\n"
             << "s : stampa\n"
             << "u : uscita\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'i':
            cout << "Valore da inserire? ";
            cin >> val;
            inserisci(q, val);
            break;
          case 'e':
            if (! elimina(q))
                cout << "Coda vuota!\n";
            else
                cout << "Eliminazione ok!\n";
            break;
          case 'p':
            if (! primo(q, val))
                cout << "Coda vuota!\n";
            else
              cout << "Primo elemento della coda: " << val << endl;
            break;
          case 's':
            cout << "Contenuto della coda: ";
            stampa(q);
            break;
          default:
            if (scelta != 'u')
                cout << "Operazione non valida: " << scelta << endl;
        }
    } while (scelta != 'u');
```

```
    return 0;
}
```

3 **queue_A34.h**

```
#ifndef QUEUE_H
 #define QUEUE_H


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

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


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


void   crea (queue &q);
void   inserisci (queue &q, float val);
retval elimina (queue &q);
retval vuota   (const queue &q);
retval primo   (const queue &q, float &result);
void   stampa   (const queue &q);


#endif // QUEUE_H
```

3 **queue_A34.cc**

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

#include "queue.h"


void crea (queue &q)
{
    q.head = q.tail = NULL;
}

retval vuota (const queue &q)
{
    return (q.head == NULL ? TRUE : FALSE);
}

void inserisci (queue &q, float val)
```

```
{
    node *n = new node;

    n->val = val;
    n->next = NULL;
    if (vuota(q))
      q.head = q.tail = n;
    else
    {
      q.tail->next = n;
      q.tail = n;
    }
}

retval elimina (queue &q)
{
    node *first;
    if (vuota(q))
      return FALSE;

    first = q.head;
    q.head = q.head->next;
    delete first;

    if (vuota(q))
      q.tail = NULL;

    return TRUE;
}

retval primo (const queue &q, float &result)
{
    if (vuota(q))
      return FALSE;

    result = q.head->val;
    return TRUE;
}


void stampa (const queue &q)
{
    node *n = q.head;
    while (n != NULL)
    {
      cout << n->val << " ";
      n = n->next;
    }
    cout << endl;
}
```