Programmazione 1

Esame completo con soluzioni del prof

8 gennaio 2014

Nel file `gennaio_2014.zip` ci sono tutti i sorgenti comodi da copiare.

Esercizio 1a

Scrivere nel file `esercizio1.cc` un programma che, preso come unico argomento della `main` il nome di un file di testo contenente una sequenza di **numeri reali** (preceduta da due interi), legga questi numeri reali dal file e li memorizzi in una matrice di **double** allocata dinamicamente.

L'allocazione dinamica della matrice nonché la lettura dei dati dal file e il popolamento della matrice stessa vanno effettuati nella funzione "`leggiMatrice`", che riceve **come parametri in ingresso lo stream di input (passato per riferimento), il numero di righe e il numero di colonne della matrice**, e che **ritorna la matrice stessa**.

Si assuma che il file sia composto da alcune righe, composte ciascuna da una sequenza di numeri reali separati dal carattere spazio (' '). **Il file comincia con due numeri interi**, corrispondenti rispettivamente **al numero di righe** e **al numero di colonne della matrice**; le altre righe corrispondono alle righe della matrice, colonna per colonna.

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

```
3 11
1.5 1.7 2.0 3.9 5.14 0.7 9.12 8.0 -14 5.19 6.7
9 8.2 5 6 7.1 12 3.6 45 6 11.5 0.7
94 95.9 100 0 -1.1 -2.3 -3.1323 0 0 0 -1
```

allora il comando:

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

porta a salvare una matrice in memoria di dimensioni "`3x11`", equivalente a quella dichiarata staticamente come:

```
{{1.5,1.7,2.0,3.9,5.14,0.7,9.12,8.0,-14,5.19,6.7}
{9,8.2,5,6,7.1,12,3.6,45,6,11.5,0.7}
{94,95.9,100,0,-1.1,-2.3,-3.1323,0,0,0,-1}}
```

NOTA 1: Si assuma che il file contenga il corretto numero di elementi, conformemente al numero di righe e di colonne dichiarato all'inizio del file stesso.

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

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

esercizio1.cc

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

using namespace std;

// Dichiarazione della funzione "leggiMatrice"
double** leggiMatrice(fstream& in, int r, int c);

int main(int argc, char * argv[]) {
  fstream in;
  int righe, colonne, n;

  // Controllo argomenti passati in ingresso
  if (argc != 2) {
    cerr << "Sintassi: ./a.out <in>." << endl;
    exit(EXIT_FAILURE);
  }

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

  // Legge il numero di righe e delle colonne necessarie
  // dai primi due numeri contenuti nel file
  in >> righe >> colonne;

  // Invoca la funzione "leggiMatrice"
  double** matrice = leggiMatrice(in, righe, colonne);

  // Chiude il file
  in.close();

  return 0;
}

// Implementazione della funzione "leggiMatrice"
double** leggiMatrice(fstream& in, int r, int c) {
  int i;
  double d;

  // Alloca la matrice dinamicamente
  double** m = new double* [r];
  for(i = 0; i < r; i++) {
    m[i] = new double[c];
  }

  // Legge i dati dal file
  i = 0;
  in >> d;
  while (!in.eof()) {
```

```
    m[i / c][i % c] = d;
    i++;
    in >> d;
  }

  // Ritorna la matrice
  return m;
}
```

Esercizio 1b

Scrivere nel file `esercizio1.cc` un programma che, preso come unico argomento della `main` il nome di un file di testo contenente una sequenza di **caratteri** (preceduta da due interi), legga questi caratteri dal file e li memorizzi in una matrice di **char** allocata dinamicamente.

L'allocazione dinamica della matrice nonché la lettura dei dati dal file e il popolamento della matrice stessa vanno effettuati nella funzione "`leggiMatrice`", che riceve **come parametri in ingresso lo stream di input (passato per riferimento), il numero di righe e il numero di colonne della matrice**, e che **ritorna la matrice stessa**.

Si assuma che il file sia composto da alcune righe, composte ciascuna da una sequenza di caratteri separati dal carattere spazio (' '). **Il file comincia con due numeri interi**, corrispondenti rispettivamente **al numero di righe** e **al numero di colonne della matrice**; le altre righe corrispondono alle righe della matrice, colonna per colonna.

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

```
7 4
a & c d
f g h j
k l + n
q r s t
a w 1 t
z p n !
0 4 r %
```

allora il comando:

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

porta a salvare una matrice in memoria di dimensioni "`7x4`", equivalente a quella dichiarata staticamente come:

```
{{'a','&','c','d'},
{'f','g','h','j'},
{'k','l','+','n'},
{'q','r','s','t'},
{'a','w','1','t'},
{'z','p','n','!'},
{'0','4','r','%'}}
```

NOTA 1: Si assuma che il file contenga il corretto numero di elementi, conformemente al numero di righe e di colonne dichiarato all'inizio del file stesso.

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

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

esercizio1.cc

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

using namespace std;

// Dichiarazione della funzione "leggiMatrice"
char** leggiMatrice(fstream& in, int r, int c);

int main(int argc, char * argv[]) {
  fstream in;
  int righe, colonne, n;

  // Controllo argomenti passati in ingresso
  if (argc != 2) {
    cerr << "Sintassi: ./a.out <in>." << endl;
    exit(EXIT_FAILURE);
  }

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

  // Legge il numero di righe e delle colonne necessarie
  // dai primi due numeri contenuti nel file
  in >> righe >> colonne;

  // Invoca la funzione "leggiMatrice"
  char** matrice = leggiMatrice(in, righe, colonne);

  // Chiude il file
  in.close();

  return 0;
}

// Implementazione della funzione "leggiMatrice"
char** leggiMatrice(fstream& in, int r, int c) {
  int i;
  char car;

  // Alloca la matrice dinamicamente
  char** m = new char* [r];
  for(i = 0; i < r; i++) {
    m[i] = new char[c];
  }

  // Legge i dati dal file
  i = 0;
  in >> car;
  while (!in.eof()) {
```

```
    m[i / c][i % c] = car;
    i++;
    in >> car;
  }

  // Ritorna la matrice
  return m;
}
```

Esercizio 1c

Scrivere nel file `esercizio1.cc` un programma che, preso come unico argomento della `main` il nome di un file di testo contenente una sequenza di **numeri reali** (preceduta da due interi), legga questi numeri reali dal file e li memorizzi in una matrice di **double** allocata dinamicamente.

L'allocazione dinamica della matrice nonché la lettura dei dati dal file e il popolamento della matrice stessa vanno effettuati nella funzione "`leggiMatrice`", che riceve **come parametri in ingresso lo stream di input (passato per riferimento), il numero di righe e il numero di colonne della matrice**, e che **ritorna la matrice stessa**.

Si assuma che il file sia composto da alcune righe, composte ciascuna da una sequenza di numeri reali separati dal carattere spazio (' '). **Il file comincia con due numeri interi**, corrispondenti rispettivamente **al numero di colonne** e **al numero di righe della matrice**; le altre righe corrispondono alle righe della matrice, colonna per colonna.

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

```
7 5
6 9.9 12 37 8 5.5 0
15 0.001 7 14 43.16 1 0
1 2 3 -3 -2 11 8
11 11 11 10.9876 99 3 4
88 5.14 3.12 0.9 15 7 12
```

allora il comando:

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

porta a salvare una matrice in memoria di dimensioni "`5x7`", equivalente a quella dichiarata staticamente come:

```
{{6,9.9,12,37,8,5.5,0},
{15,0.001,7,14,43.16,1,0},
{1,2,3,-3,-2,11,8},
{11,11,11,10.9876,99,3,4},
{88,5.14,3.12,0.9,15,7,12}}
```

NOTA 1: Si assuma che il file contenga il corretto numero di elementi, conformemente al numero di righe e di colonne dichiarato all'inizio del file stesso.

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

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

esercizio1.cc

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

using namespace std;

// Dichiarazione della funzione "leggiMatrice"
double** leggiMatrice(fstream& in, int r, int c);

int main(int argc, char * argv[]) {
  fstream in;
  int righe, colonne, n;

  // Controllo argomenti passati in ingresso
  if (argc != 2) {
    cerr << "Sintassi: ./a.out <in>." << endl;
    exit(EXIT_FAILURE);
  }

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

  // Legge il numero di righe e delle colonne necessarie
  // dai primi due numeri contenuti nel file
  in >> colonne >> righe;

  // Invoca la funzione "leggiMatrice"
  double** matrice = leggiMatrice(in, righe, colonne);

  // Chiude il file
  in.close();

  return 0;
}

// Implementazione della funzione "leggiMatrice"
double** leggiMatrice(fstream& in, int r, int c) {
  int i;
  double d;

  // Alloca la matrice dinamicamente
  double** m = new double* [r];
  for(i = 0; i < r; i++) {
    m[i] = new double[c];
  }

  // Legge i dati dal file
  i = 0;
  in >> d;
  while (!in.eof()) {
```

```
    m[i / c][i % c] = d;
    i++;
    in >> d;
  }

  // Ritorna la matrice
  return m;
}
```

Esercizio 1d

Scrivere nel file `esercizio1.cc` un programma che, preso come unico argomento della `main` il nome di un file di testo contenente una sequenza di **caratteri** (preceduta da due interi), legga questi caratteri dal file e li memorizzi in una matrice di **char** allocata dinamicamente.

L'allocazione dinamica della matrice nonché la lettura dei dati dal file e il popolamento della matrice stessa vanno effettuati nella funzione "`leggiMatrice`", che riceve **come parametri in ingresso lo stream di input (passato per riferimento), il numero di righe e il numero di colonne della matrice**, e che **ritorna la matrice stessa**.

Si assuma che il file sia composto da alcune righe, composte ciascuna da una sequenza di caratteri separati dal carattere spazio (' '). **Il file comincia con due numeri interi**, corrispondenti rispettivamente **al numero di colonne** e **al numero di righe della matrice**; le altre righe corrispondono alle righe della matrice, colonna per colonna.

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

```
12 3
* 0 * 7 a b c d l l l 2
! % 1 t ~ - 9 4 r t s q
2 + 2 = 4 a w q r g b n
```

allora il comando:

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

porta a salvare una matrice in memoria di dimensioni "`3x12`", equivalente a quella dichiarata staticamente come:

```
{{'*','0','*','7','a','b','c','d','l','l','l','2'},
{'!','%','1','t','~','-','9','4','r','t','s','q'},
{'2','+','2','=','4','a','w','q','r','g','b','n'}}
```

NOTA 1: Si assuma che il file contenga il corretto numero di elementi, conformemente al numero di righe e di colonne dichiarato all'inizio del file stesso.

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

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

esercizio1.cc

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

using namespace std;

// Dichiarazione della funzione "leggiMatrice"
char** leggiMatrice(fstream& in, int r, int c);

int main(int argc, char * argv[]) {
  fstream in;
  int righe, colonne, n;

  // Controllo argomenti passati in ingresso
  if (argc != 2) {
    cerr << "Sintassi: ./a.out <in>." << endl;
    exit(EXIT_FAILURE);
  }

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

  // Legge il numero di righe e delle colonne necessarie
  // dai primi due numeri contenuti nel file
  in >> colonne >> righe;

  // Invoca la funzione "leggiMatrice"
  char** matrice = leggiMatrice(in, righe, colonne);

  // Chiude il file
  in.close();

  return 0;
}

// Implementazione della funzione "leggiMatrice"
char** leggiMatrice(fstream& in, int r, int c) {
  int i;
  char car;

  // Alloca la matrice dinamicamente
  char** m = new char* [r];
  for(i = 0; i < r; i++) {
    m[i] = new char[c];
  }

  // Legge i dati dal file
  i = 0;
  in >> car;
  while (!in.eof()) {
```

```
    m[i / c][i % c] = car;
    i++;
    in >> car;
  }

  // Ritorna la matrice
  return m;
}
```

Esercizio 2a

Scrivere nel file `esercizio2.cc` un programma che simuli il lancio di due dadi per volta. Ogni dado ha 6 facce. Il programma termina dopo aver effettuato 5 lanci o se il risultato dell'ultimo lancio è pari a 12 (cioè il massimo punteggio ottenibile con due dadi).

Per ogni lancio il programma deve stampare il risultato di ogni dado e la somma dei loro valori, come negli esempi di esecuzione riportati di seguito:

Caso 1: il programma termina dopo il numero massimo di tiri

```
Lancio 1: dado1=4, dado2=5, somma=9
Lancio 2: dado1=1, dado2=3, somma=4
Lancio 3: dado1=4, dado2=5, somma=9
Lancio 4: dado1=6, dado2=2, somma=8
Lancio 5: dado1=1, dado2=3, somma=4
Il programma termina al lancio numero 5 con un punteggio massimo di 9
```

Caso 2: il programma termina perchè si e' ottenuto il punteggio massimo

```
Lancio 1: dado1=5, dado2=4, somma=9
Lancio 2: dado1=6, dado2=6, somma=12
Il programma termina al lancio numero 2 con un punteggio massimo di 12
```

NOTA 1: Va implementata solamente la funzione `void gioca(...)`

NOTA 2: All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria diverse da quelle già presenti.

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

esercizio2.cc

```
using namespace std;

#include <iostream>      /* cout */
#include <stdlib.h>      /* srand, rand */
#include <time.h>        /* time */

// definizione delle funzioni
int tira_dado();
void gioca(int &, int &);

int main () {
  // init random seed
  srand (time(NULL));

  int num_lanci, somma_max;
  gioca(num_lanci, somma_max);
  cout << "Il programma termina al lancio numero "
       << num_lanci << " con un punteggio massimo di "
       << somma_max << endl;

  return(0);
}

int tira_dado() {
  return 1 + rand() % 6;
}

void gioca(int &i, int &max) {
  int somma=0;
  max=0;
  for (i = 0; i<5 && somma<12; i++) {
    int dado1 = tira_dado();
    int dado2 = tira_dado();
    somma = dado1 + dado2;
    cout << "Lancio " << i+1 << ": dado1="
         << dado1 << ", dado2=" << dado2
         << ", somma=" << somma << endl;
    if (somma > max) {
       max = somma;
    }
  }
}
```

Esercizio 2b

Scrivere nel file `esercizio2.cc` un programma che simuli il lancio di due dadi per volta. Ogni dado ha 4 facce. Il programma termina dopo aver effettuato 5 lanci o se il risultato dell'ultimo lancio è pari a 8 (cioè il massimo punteggio ottenibile con due dadi).

Per ogni lancio il programma deve stampare il risultato di ogni dado e la somma dei loro valori, come negli esempi di esecuzione riportati di seguito:

Caso 1: il programma termina dopo il numero massimo di tiri

```
Lancio 1: dado1=4, dado2=2, somma=6
Lancio 2: dado1=1, dado2=3, somma=4
Lancio 3: dado1=4, dado2=1, somma=5
Lancio 4: dado1=4, dado2=2, somma=6
Lancio 5: dado1=1, dado2=2, somma=3
Il programma termina al lancio numero 5 con un punteggio massimo di 6
```

Caso 2: il programma termina perchè si e' ottenuto il punteggio massimo

```
Lancio 1: dado1=4, dado2=4, somma=8
Il programma termina al lancio numero 1 con un punteggio massimo di 8
```

NOTA 1: Va implementata solamente la funzione `void gioca(...)`

NOTA 2: All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria diverse da quelle già presenti.

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

esercizio2.cc

```
using namespace std;

#include <iostream>      /* cout */
#include <stdlib.h>      /* srand, rand */
#include <time.h>        /* time */

// definizione delle funzioni
int tira_dado();
void gioca(int &, int &);

int main () {
  // init random seed
  srand (time(NULL));

  int num_lanci, somma_max;
  gioca(num_lanci, somma_max);
  cout << "Il programma termina al lancio numero "
       << num_lanci << " con un punteggio massimo di "
       << somma_max << endl;

  return(0);
}

int tira_dado() {
  return 1 + rand() % 4;
}

void gioca(int &i, int &max) {
  int somma=0;
  max=0;
  for (i = 0; i<5 && somma<8; i++) {
    int dado1 = tira_dado();
    int dado2 = tira_dado();
    somma = dado1 + dado2;
    cout << "Lancio " << i+1 << ": dado1="
         << dado1 << ", dado2=" << dado2
         << ", somma=" << somma << endl;
    if (somma > max) {
       max = somma;
    }
  }
}
```

Esercizio 2c

Scrivere nel file `esercizio2.cc` un programma che simuli il lancio di due dadi per volta. Ogni dado ha 6 facce. Il programma termina dopo aver effettuato 10 lanci o se il risultato dell'ultimo lancio è pari a 12 (cioè il massimo punteggio ottenibile con due dadi).

Per ogni lancio il programma deve stampare il risultato di ogni dado e la somma dei loro valori, come negli esempi di esecuzione riportati di seguito:

Caso 1: il programma termina dopo il numero massimo di tiri

```
Lancio 1: dado1=3, dado2=3, somma=6
Lancio 2: dado1=1, dado2=1, somma=2
Lancio 3: dado1=3, dado2=3, somma=6
Lancio 4: dado1=2, dado2=3, somma=5
Lancio 5: dado1=2, dado2=2, somma=4
Lancio 6: dado1=2, dado2=4, somma=6
Lancio 7: dado1=4, dado2=2, somma=6
Lancio 8: dado1=1, dado2=4, somma=5
Lancio 9: dado1=2, dado2=3, somma=5
Lancio 10: dado1=5, dado2=6, somma=11
Il programma termina al lancio numero 10 con un punteggio massimo di 11
```

Caso 2: il programma termina perchè si e' ottenuto il punteggio massimo

```
Lancio 1: dado1=5, dado2=5, somma=10
Lancio 2: dado1=6, dado2=6, somma=12
Il programma termina al lancio numero 2 con un punteggio massimo di 12
```

NOTA 1: Va implementata solamente la funzione `void gioca(...)`

NOTA 2: All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria diverse da quelle già presenti.

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

esercizio2.cc

```
using namespace std;

#include <iostream>      /* cout */
#include <stdlib.h>      /* srand, rand */
#include <time.h>        /* time */

// definizione delle funzioni
int tira_dado();
void gioca(int &, int &);

int main () {
  // init random seed
  srand (time(NULL));

  int num_lanci, somma_max;
  gioca(num_lanci, somma_max);
  cout << "Il programma termina al lancio numero "
       << num_lanci << " con un punteggio massimo di "
       << somma_max << endl;

  return(0);
}

int tira_dado() {
  return 1 + rand() % 6;
}

void gioca(int &i, int &max) {
  int somma=0;
  max=0;
  for (i = 0; i<10 && somma<12; i++) {
    int dado1 = tira_dado();
    int dado2 = tira_dado();
    somma = dado1 + dado2;
    cout << "Lancio " << i+1 << ": dado1="
         << dado1 << ", dado2=" << dado2
         << ", somma=" << somma << endl;
    if (somma > max) {
       max = somma;
    }
  }
}
```

Esercizio 2d

Scrivere nel file `esercizio2.cc` un programma che simuli il lancio di due dadi per volta. Ogni dado ha 4 facce. Il programma termina dopo aver effettuato 10 lanci o se il risultato dell'ultimo lancio è pari a 8 (cioè il massimo punteggio ottenibile con due dadi).

Per ogni lancio il programma deve stampare il risultato di ogni dado e la somma dei loro valori, come negli esempi di esecuzione riportati di seguito:

Caso 1: il programma termina dopo il numero massimo di tiri

```
Lancio 1: dado1=2, dado2=1, somma=3
Lancio 2: dado1=1, dado2=2, somma=3
Lancio 3: dado1=4, dado2=3, somma=7
Lancio 4: dado1=2, dado2=2, somma=4
Lancio 5: dado1=4, dado2=1, somma=5
Lancio 6: dado1=4, dado2=1, somma=5
Lancio 7: dado1=2, dado2=2, somma=4
Lancio 8: dado1=1, dado2=3, somma=4
Lancio 9: dado1=1, dado2=3, somma=4
Lancio 10: dado1=3, dado2=3, somma=6
Il programma termina al lancio numero 10 con un punteggio massimo di 7
```

Caso 2: il programma termina perchè si e' ottenuto il punteggio massimo

```
Lancio 1: dado1=4, dado2=4, somma=8
Il programma termina al lancio numero 1 con un punteggio massimo di 8
```

NOTA 1: Va implementata solamente la funzione `void gioca(...)`

NOTA 2: All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria diverse da quelle già presenti.

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

esercizio2.cc

```
using namespace std;

#include <iostream>      /* cout */
#include <stdlib.h>      /* srand, rand */
#include <time.h>        /* time */

// definizione delle funzioni
int tira_dado();
void gioca(int &, int &);

int main () {
  // init random seed
  srand (time(NULL));

  int num_lanci, somma_max;
  gioca(num_lanci, somma_max);
  cout << "Il programma termina al lancio numero "
       << num_lanci << " con un punteggio massimo di "
       << somma_max << endl;

  return(0);
}

int tira_dado() {
  return 1 + rand() % 4;
}

void gioca(int &i, int &max) {
  int somma=0;
  max=0;
  for (i = 0; i<10 && somma<8; i++) {
    int dado1 = tira_dado();
    int dado2 = tira_dado();
    somma = dado1 + dado2;
    cout << "Lancio " << i+1 << ": dado1="
         << dado1 << ", dado2=" << dado2
         << ", somma=" << somma << endl;
    if (somma > max) {
       max = somma;
    }
  }
}
```

Esercizio 3a

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).

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;
}
```

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
```

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)
{
    if (is_full(q)) {
        return false;
    }
    q.elem[q.tail] = n;
    q.tail = next(q.tail, q);
    return true;
}

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

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

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

Esercizio 3b

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).

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;
}
```

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
```

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)
{
    if (piena(q)) {
        return false;
    }
    q.elem[q.tail] = n;
    q.tail = next(q.tail, q);
    return true;
}

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

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

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

Esercizio 3c

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).

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;
}
```

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
```

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)
{
    if (is_full(q)) {
        return false;
    }
    q.elem[q.tail] = n;
    q.tail = next(q.tail, q);
    return true;
}

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

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

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

Esercizio 3d

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).

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;
}
```

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
```

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)
{
    if (piena(q)) {
        return false;
    }
    q.elem[q.tail] = n;
    q.tail = next(q.tail, q);
    return true;
}

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

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

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

Esercizio 4

Si definisca una funzione **ricorsiva** "`palindroma`" che prenda in ingresso due stringhe "*stringa*" e "*err*" e restituisca un valore booleano che vale VERO qualora la stringa passata in ingresso sia palindroma, FALSO altrimenti. Se la stringa "*stringa*" risulta parzialmente palindroma, la parte di essa che non soddisfa il criterio specificato va ritornata mediante la stringa "*err*".

Definizioni:

1. una stringa si dice palindroma se è simmetrica rispetto al suo centro, ovvero se può essere letta indifferentemente da sinistra verso destra o da destra verso sinistra;
2. caratteri maiuscoli e minuscoli sono considerati equivalenti al fine di determinare se una stringa è palindroma o meno (es.: "aDa" è palindroma come "aDA");
3. caratteri non alfanumerici non vanno considerati nella verifica di cui al punto (a) (es.: "Amore, Roma." è palindroma).

Esempio di esecuzione:

```
Immetti un'altra frase che ritieni palindroma: E d'Irene se ne ride!
La frase 'E d'Irene se ne ride!' e' palindroma

Immetti un'altra frase che ritieni palindroma: Programmazione
La sotto-frase 'Programmazione' non e' palindroma
```

NOTA 1: Non è consentito utilizzare alcuna forma di ciclo.

NOTA 2: È consentito definire e utilizzare eventuali funzioni ausiliarie, purché a loro volta ricorsive e senza cicli.

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

esercizio4.cc

```
#include <iostream>
#include <cstring>

using namespace std;

const int SIZE = 1000;

char * palindroma1(char * stringa, int l, int r);
bool palindroma(char * stringa, char *& err);

int main() {
  int n;
  char a[SIZE];
  char * b;

  do {
    if (palindroma(a, b))
      cout << "La frase `" << a << "' e` palindroma\n";
    else
      cout << "La sotto-frase `" << b << "' non e` palindroma\n";
    cout << "\nImmetti un'altra frase che ritieni palindroma: ";
    cin.getline(a,SIZE);
  } while (a[0] != '\0');

  return 0;
}

char * palindroma1(char * s, int l, int r) {
  char * res;
  char lc = s[l];
  char rc = s[r];

  if (r-l<=1) {
    res = NULL;
  } else if (!isalpha(lc)) {
    res = palindroma1(s, l+1, r);
  } else if (!isalpha(rc)) {
    res = palindroma1(s, l, r-1);
  } else if (rc==lc || rc==lc-'A'+'a' || rc==lc+'A'-'a') {
    res = palindroma1(s, l+1, r-1);
  } else {
    res = new char[r-l+2];
    strncpy(res, &s[l], r-l+1);
    strcat(res,"\0");
  }
  return res;
}


bool palindroma(char * stringa, char *& err) {
  err = palindroma1(stringa, 0, strlen(stringa) - 1);
  return (err == NULL);
}
```