Primo Appello di Programmazione I

12 Gennaio 2011
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 Sia `input1.txt` un file di testo contenente una sequenza di stringhe formate da ripetizioni dell'unico carattere `'*'`, separate tra loro da uno o più spazi e ritorni a capo. Le stringhe rappresentano una sequenza di interi positivi codificati in codice unario (cioè la lunghezza di ogni stringa rappresenta il valore associato). Ad esempio, il seguente file

```
****** *** ******** **
***** ******** *************
**** ***** ****
** ******** *************
**** ******* *****
******* * *************
******** **** ****************
******* ************
```

corrisponde alla sequenza di interi 6 3 8 2 5 8 13 4 5 4 2 8 13 4 7 5 7 1 13 8 4 16 7 12.

Nel file `esercizio1.cpp` è definita la funzione `main` che invoca la funzione `leggiECalcolaMedia`, passando il nome del file `input1.txt`, e stampa il risultato. Scrivere nel file `esercizio1.cpp` la definizione della funzione `leggiECalcolaMedia`, la quale calcola la media dei valori contenuti nel file ed appende ad esso il risultato rappresentato in codice unario (ed arrotondato per eccesso). Nell'esempio precedente, dopo l'invocazione della funzione `leggiECalcolaMedia`, il file deve essere il seguente:

```
****** *** ******** **
***** ******** *************
**** ***** ****
** ******** *************
**** ******* *****
******* * *************
******** **** ****************
******* ************
*******
```

in cui il valore 7 (cioè la media arrotondata per eccesso dei valori 6 3 8 2 5 8 13 4 5 4 2 8 13 4 7 5 7 1 13 8 4 16 7 12) è stato appeso alla fine del file. Inoltre, `leggiECalcolaMedia` memorizza il valore della media nella variabile di tipo *float* passata per riferimento.

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

1 **esercizio1.cc**

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


void leggiECalcolaMedia(char nome_file[], float &media);


int main () {

   char nomeFile[] = "input1.txt";
   float result = 0;

   leggiECalcolaMedia(nomeFile, result);

   cout << "La media calcolata in " << nomeFile << " e': " << result << endl;

}


// Inserire qui la definizione della funzione leggiECalcolaMedia

void leggiECalcolaMedia(char nome_file[], float &media){

   fstream fin, fout;
   char parola[200];
   int counter = 0;

   media = 0;

   fin.open(nome_file, ios::in);

   if (fin.fail()) {
      cout << "Errore nell'apertura del file "<< nome_file << "!" << endl;
      return;
   }

  fin >> parola;

   while (!fin.eof()) { // finche lo stream non ha raggiunto la fine del file

      counter++;
      media += strlen(parola);

      fin >> parola;
   }

   fin.close();

   if (counter != 0) {
```

```
      media /= counter;

      fout.open(nome_file, ios::out | ios::app);

      if (fout.fail()) {
         cout << "Errore nell'apertura del file "<< nome_file << "!" << endl;
         return;
      }

      for(int i=0; i<media; i++){
         fout << "*";
      }
      fout << " ";

      fout.close();
   }
}
```

1 Sia `input1.txt` un file di testo contenente una sequenza di stringhe formate da ripetizioni dell'unico carattere `'+'`, separate tra loro da uno o più spazi e ritorni a capo. Le stringhe rappresentano una sequenza di interi positivi codificati in codice unario (cioè la lunghezza di ogni stringa rappresenta il valore associato). Ad esempio, il seguente file

```
++++++ +++ ++++++++++
++++++ ++++++++++++++++++++++
++++++++++ +++++++++ ++
++++++++ +++++++++++++++ +++++++++
++++ +++++++ +++++ +++++++
+ ++++++++++++
++++++++ ++++ ++++++++++++++++
```

corrisponde alla sequenza di interi 6 3 11 6 23 12 9 2 8 15 9 4 7 5 7 1 13 8 4 16.

Nel file `esercizio1.cpp` è definita la funzione `main` che invoca la funzione `leggiECalcolaMassimo`, passando il nome del file `input1.txt`, e stampa il risultato. Scrivere nel file `esercizio1.cpp` la definizione della funzione `leggiECalcolaMassimo`, la quale calcola il massimo tra i valori contenuti nel file ed appende ad esso tale valore rappresentato in codice unario. Nell'esempio precedente, dopo l'invocazione della funzione `leggiECalcolaMassimo`, il file deve essere il seguente:

```
++++++ +++ ++++++++++
++++++ ++++++++++++++++++++++
++++++++++ +++++++++ ++
++++++++ +++++++++++++++ +++++++++
++++ +++++++ +++++ +++++++
+ ++++++++++++
++++++++ ++++ ++++++++++++++++
++++++++++++++++++++++
```

in cui il valore 23 (cioè il massimo tra i valori 6 3 11 6 23 12 9 2 8 15 9 4 7 5 7 1 13 8 4 16) è stato appeso alla fine del file. Inoltre, `leggiECalcolaMassimo` memorizza il valore massimo trovato nella variabile di tipo *int* passata per riferimento.

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

1 **esercizio1.cc**

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


void leggiECalcolaMassimo(char nome_file[], int &massimo);

int main () {

   char nomeFile[] = "input1.txt";
   int result = 0;

   leggiECalcolaMassimo(nomeFile, result);

   cout << "Il numero massimo in " << nomeFile << " e': " << result << endl;

}


// Inserire qui la definizione della funzione leggiECalcolaMassimo

void leggiECalcolaMassimo(char nome_file[], int &massimo){

   fstream fin, fout;
   char parola[200];

   int numero, counter = 0;
   massimo = 0;

   fin.open(nome_file, ios::in);

   if (fin.fail()) {
      cout << "Errore nell'apertura del file "<< nome_file << "!" << endl;
      return;
   }

  fin >> parola;

   while (!fin.eof()) { // finche lo stream non ha raggiunto la fine del file

      counter++;
      numero = strlen(parola);

      if (numero > massimo) {
         massimo = numero;
      }

      fin >> parola;
   }

   fin.close();
```

```
   if (counter != 0) {

      fout.open(nome_file, ios::out | ios::app);

      if (fout.fail()) {
         cout << "Errore nell'apertura del file "<< nome_file << "!" << endl;
         return;
      }

      for(int i=0; i<massimo; i++){
         fout << "+";
      }
      fout << " ";

      fout.close();
   }
}
```

1 Sia `input1.txt` un file di testo contenente una sequenza di stringhe formate da ripetizioni dell'unico carattere `'#'`, separate tra loro da uno o più spazi e ritorni a capo. Le stringhe rappresentano una sequenza di interi positivi codificati in codice unario (cioè la lunghezza di ogni stringa rappresenta il valore associato). Ad esempio, il seguente file

```
###### ### ######## ######
############## ##### ####
########## ########## ## ########
###### ############# #####
#### ############### ##### #######
####### ######
######## #### ########
```

corrisponde alla sequenza di interi 6 3 8 6 14 5 4 11 10 2 8 6 13 5 4 15 5 7 7 6 8 4 8.

Nel file `esercizio1.cpp` è definita la funzione `main` che invoca la funzione `leggiECalcolaMinimo`, passando il nome del file `input1.txt`, e stampa il risultato. Scrivere nel file `esercizio1.cpp` la definizione della funzione `leggiECalcolaMinimo`, la quale calcola il minimo tra i valori contenuti nel file ed appende ad esso tale valore rappresentato in codice unario. Nell'esempio precedente, dopo l'invocazione della funzione `leggiECalcolaMinimo`, il file deve essere il seguente:

```
###### ### ######## ######
############## ##### ####
########## ########## ## ########
###### ############# #####
#### ############### ##### #######
####### ######
######## #### ########
##
```

in cui il valore 2 (cioè il minimo tra i valori 6 3 8 6 14 5 4 11 10 2 8 6 13 5 4 15 5 7 7 6 8 4 8) è stato appeso alla fine del file. Inoltre, `leggiECalcolaMinimo` memorizza il valore minimo trovato nella variabile di tipo *int* passata per riferimento.

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

1 **esercizio1.cc**

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


void leggiECalcolaMinimo(char nome_file[], int &minimo);

int main () {

   char nomeFile[] = "input1.txt";
   int result = 0;

   leggiECalcolaMinimo(nomeFile, result);

   cout << "Il numero minimo in " << nomeFile << " e': " << result << endl;

}


// Inserire qui la definizione della  funzione leggiECalcolaMinimo

void leggiECalcolaMinimo(char nome_file[], int &minimo){

   fstream fin, fout;
   char parola[200];

   int numero, counter = 0;
   minimo = 200;

   fin.open(nome_file, ios::in);

   if (fin.fail()) {
      cout << "Errore nell'apertura del file "<< nome_file << "!" << endl;
      return;
   }

  fin >> parola;

   while (!fin.eof()) { // finche lo stream non ha raggiunto la fine del file

      counter++;
      numero = strlen(parola);

      if (minimo > numero) {
         minimo = numero;
      }

      fin >> parola;
   }

   fin.close();
```

```
   if (counter != 0) {

      fout.open(nome_file, ios::out | ios::app);

      if (fout.fail()) {
         cout << "Errore nell'apertura del file "<< nome_file << "!" << endl;
         return;
      }

      for(int i=0; i<minimo; i++){
         fout << "#";
      }
      fout << " ";

      fout.close();
   }
}
```

1 Sia `input1.txt` un file di testo contenente una sequenza di stringhe formate da ripetizioni dell'unico carattere `'%'`, separate tra loro da uno o più spazi e ritorni a capo. Le stringhe rappresentano una sequenza di interi positivi codificati in codice unario (cioè la lunghezza di ogni stringa rappresenta il valore associato). Ad esempio, il seguente file

```
%%%% %%% %%%%%%% %%%%%%
%%%%% %% %%%%%%%%
%%%%% %%%%%%% %
%%%%%%%% %%%%
%% %%%%%%%%%%
```

corrisponde alla sequenza di interi 4 3 7 6 5 2 8 5 7 1 8 4 2 10.

Nel file `esercizio1.cpp` è definita la funzione `main` che invoca la funzione `leggiECalcolaSomma`, passando il nome del file `input1.txt`, e stampa il risultato. Scrivere nel file `esercizio1.cpp` la definizione della funzione `leggiECalcolaSomma`, la quale calcola la somma dei valori contenuti nel file ed appende ad esso il risultato rappresentato in codice unario. Nell'esempio precedente, dopo l'invocazione della funzione `leggiECalcolaSomma`, il file deve essere il seguente:

```
%%%% %%% %%%%%%% %%%%%%
%%%%% %% %%%%%%%%
%%%%% %%%%%%% %
%%%%%%%% %%%%
%% %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

in cui il valore 72 (cioè la somma dei valori 4 3 7 6 5 2 8 5 7 1 8 4 2 10) è stato appeso alla fine del file. Inoltre, `leggiECalcolaSomma` memorizza il risultato nella variabile di tipo *int* passata per riferimento.

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

1 **esercizio1.cc**

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


void leggiECalcolaSomma(char nome_file[], int &somma);

int main () {

   char nomeFile[] = "input1.txt";
   int result = 0;

   leggiECalcolaSomma(nomeFile, result);

   cout << "La somma dei numeri in " << nomeFile << " e': " << result << endl;

}


// Inserire qui la definizione della funzione leggiECalcolaSomma

void leggiECalcolaSomma(char nome_file[], int &somma){

   fstream fin, fout;
   char parola[200];

   int counter = 0;
   somma = 0;

   fin.open(nome_file, ios::in);

   if (fin.fail()) {
      cout << "Errore nell'apertura del file "<< nome_file << "!" << endl;
      return;
   }

  fin >> parola;

   while (!fin.eof()) { // finche lo stream non ha raggiunto la fine del file

      counter++;
      somma += strlen(parola);

      fin >> parola;
   }

   fin.close();

   if (counter != 0) {

      fout.open(nome_file, ios::out | ios::app);
```

```
      if (fout.fail()) {
         cout << "Errore nell'apertura del file "<< nome_file << "!" << endl;
         return;
      }

      for(int i=0; i<somma; i++){
         fout << "%";
      }
      fout << " ";

      fout.close();
   }
}
```

2 Scrivere nel file `esercizio1.cc` la dichiarazione e la definizione della funzione `array_palindromo` che verifica in modo **ricorsivo** se una sequenza di numeri contenuta in un array di al più 100 elementi di tipo `int` è palindroma. Il contenuto di tale array è considerato palindromo se il primo elemento è uguale all'ultimo, il secondo elemento è uguale al penultimo, e così via... per tutti gli elementi dell'array.

Ad esempio il seguente array:

```
9 22 5 5 22 9
```

è palindromo in quanto il primo elemento (`9`) è uguale all'ultimo, il secondo elemento (`22`) è uguale al penultimo, ed infine il terzo elemento (`5`) è uguale al terzultimo. Anche il seguente array di 5 elementi è palindromo:

```
0 8 33 8 0
```

Infatti, in caso di numero dispari di elementi, l'elemento centrale è ininfluente.
Invece, l'array:

```
10 2 7 19 4 0 4 18 7 2 10
```

non è palindromo, infatti il quarto numero a partire dalla testa dell'array (`19`) è diverso dal quarto numero a partire dal fondo (ovvero `18`).

NOTA: **all'interno della funzione `array_palindromo` non ci possono essere cicli o chiamate a funzioni contenenti cicli; si deve invece fare uso di funzioni ausiliarie purchè a loro volta ricorsive.**

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

2 **esercizio2.cc**

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

// Inserire qui le dichiarazioni
bool pal1 (int *sequenza, int primo, int ultimo);
bool array_palindromo (int *sequenza, int dim);

int main()
{
  int dim;
  int array[100];

  do
  {
    cout << "Inserisci il numero (0 < ... < 100) di elementi da memorizzare nell'array: ";
    cin >> dim;
  } while (dim <= 0);

  if (dim > 100)
  {
    cout << "Puoi inserire solo 100 elementi!" << endl;
    dim = 100;
  }
  for (int i = 0; i < dim; i++)
  {
    cout << "Inserire intero " << i+1 <<" : ";
    cin >> array[i];
  }

  if ( array_palindromo(array, dim) )
    cout << "La sequenza inserita e' palindroma!" << endl;
  else
    cout << "La sequenza inserita NON e' palindroma!" << endl;

  return 0;
}

// Inserire qui le definizioni

bool pal1 (int *sequenza, int primo, int ultimo)
{
  if (primo > ultimo)
    return true;
  else if (sequenza[primo] != sequenza[ultimo])
    return false;
  else
    return pal1(sequenza, primo+1, ultimo-1);
}

bool array_palindromo (int *sequenza, int dim)
{
  return pal1(sequenza, 0, dim-1);
```

```
}
```

2 Scrivere nel file `esercizio1.cc` la dichiarazione e la definizione della funzione `palindroma` che verifica in modo **ricorsivo** se una sequenza di caratteri contenuta in un array di al più 100 elementi di tipo `char` è palindroma. Il contenuto di tale array è considerato palindromo se il primo elemento è uguale all'ultimo, il secondo elemento è uguale al penultimo, e così via... per tutti gli elementi dell'array.

Ad esempio il seguente array:

```
a # 9 g F F G 9 # a
```

non è palindromo, infatti il quarto carattere a partire dalla testa dell'array (`g`) è diverso dal quarto carattere a partire dal fondo (ovvero `G`).
L'array di 5 elementi:

```
X 1 $ 1 X
```

contiene, invece, una sequenza di caratteri palindroma (si noti che, in caso di numero dispari di elementi, l'elemento centrale è ininfluente) così come è palindromo anche il contenuto dell'array:

```
m @ 4 4 @ m
```

visto che il primo elemento (`m`) è uguale all'ultimo, il secondo elemento (`@`) è uguale al penultimo, ed infine il terzo elemento (`4`) è uguale al terzultimo.

NOTA: **all'interno della funzione `palindroma` non ci possono essere cicli o chiamate a funzioni contenenti cicli; si deve invece fare uso di eventuali funzioni ausiliarie purchè a loro volta ricorsive.**

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

2 **esercizio2.cc**

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

// Inserire qui le dichiarazioni

bool pal1 (char *sequenza, int primo, int ultimo);
bool palindroma (char *sequenza, int dim);

int main()
{
  int dim;
  char array[100];

  do
  {
    cout << "Inserisci il numero (0 < ... < 100) di elementi da memorizzare nell'array: ";
    cin >> dim;
  } while (dim <= 0);

  if (dim > 100)
  {
    cout << "Puoi inserire solo 100 elementi!" << endl;
    dim = 100;
  }
  for (int i = 0; i < dim; i++)
  {
    cout << "Inserire carattere " << i+1 <<" : ";
    cin >> array[i];
  }

  if ( palindroma(array, dim) )
    cout << "La sequenza inserita e' palindroma!" << endl;
  else
    cout << "La sequenza inserita NON e' palindroma!" << endl;

  return 0;
}

// Inserire qui le definizioni

bool pal1 (char *sequenza, int primo, int ultimo)
{
  if (primo > ultimo)
    return true;
  else if (sequenza[primo] != sequenza[ultimo])
    return false;
  else
    return pal1(sequenza, primo+1, ultimo-1);
}

bool palindroma (char *sequenza, int dim)
{
```

```
  return pal1(sequenza, 0, dim-1);
}
```

2 Scrivere nel file `esercizio1.cc` la dichiarazione e la definizione della funzione `array_palindromo` che verifica in modo **ricorsivo** se una sequenza di numeri contenuta in un array di al più 100 elementi di tipo `long` è palindroma. Il contenuto di tale array è considerato palindromo se il primo elemento è uguale all'ultimo, il secondo elemento è uguale al penultimo, e così via... per tutti gli elementi dell'array.

Ad esempio il seguente array:

```
9 22 5 5 22 9
```

è palindromo in quanto il primo elemento (`9`) è uguale all'ultimo, il secondo elemento (`22`) è uguale al penultimo, ed infine il terzo elemento (`5`) è uguale al terzultimo. Anche il seguente array di 5 elementi è palindromo:

```
0 48 4333 48 0
```

Infatti, in caso di numero dispari di elementi, l'elemento centrale è ininfluente.
Invece, l'array:

```
10 2 7 19 404 0 4 18 7 2 10
```

non è palindromo, infatti il quarto numero a partire dalla testa dell'array (`19`) è diverso dal quarto numero a partire dal fondo (ovvero `18`).

NOTA: **all'interno della funzione `array_palindromo` non ci possono essere cicli o chiamate a funzioni contenenti cicli; si deve invece fare uso di funzioni ausiliarie purchè a loro volta ricorsive.**

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

2 **esercizio2.cc**

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

// Inserire qui le dichiarazioni
bool pal1 (long *sequenza, int primo, int ultimo);
bool array_palindromo (long *sequenza, int dim);

int main()
{
  int dim;
  long array[100];

  do
  {
    cout << "Inserisci il numero (0 < ... < 100) di elementi da memorizzare nell'array: ";
    cin >> dim;
  } while (dim <= 0);

  if (dim > 100)
  {
    cout << "Puoi inserire solo 100 elementi!" << endl;
    dim = 100;
  }
  for (int i = 0; i < dim; i++)
  {
    cout << "Inserire intero " << i+1 <<" : ";
    cin >> array[i];
  }

  if ( array_palindromo(array, dim) )
    cout << "La sequenza inserita e' palindroma!" << endl;
  else
    cout << "La sequenza inserita NON e' palindroma!" << endl;

  return 0;
}

// Inserire qui le definizioni

bool pal1 (long *sequenza, int primo, int ultimo)
{
  if (primo > ultimo)
    return true;
  else if (sequenza[primo] != sequenza[ultimo])
    return false;
  else
    return pal1(sequenza, primo+1, ultimo-1);
}

bool array_palindromo (long *sequenza, int dim)
{
  return pal1(sequenza, 0, dim-1);
```

```
}
```

2 Scrivere nel file `esercizio1.cc` la dichiarazione e la definizione della funzione `palindroma` che verifica in modo **ricorsivo** se una sequenza di numeri reali contenuta in un array di al più 100 elementi di tipo `double` è palindroma. Il contenuto di tale array è considerato palindromo se il primo elemento è uguale all'ultimo, il secondo elemento è uguale al penultimo, e così via... per tutti gli elementi dell'array.

Ad esempio il seguente array:

```
0.1 9 87.6 777 9 0.1
```

non è palindromo, infatti il terzo elemento a partire dalla testa dell'array (`87.6`) è diverso dal terzo elemento a partire dal fondo (ovvero `777`).
L'array di 5 elementi:

```
9.4 6.9 7 6.9 9.4
```

contiene, invece, una sequenza di numeri reali palindroma (si noti che, in caso di numero dispari di elementi, l'elemento centrale è ininfluente) così come è palindromo anche il contenuto dell'array:

```
145.2 7 7 145.2
```

NOTA: **all'interno della funzione** `palindroma` **non ci possono essere cicli o chiamate a funzioni contenenti cicli; si deve invece fare uso di eventuali funzioni ausiliarie purchè a loro volta ricorsive.**

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

2 **esercizio2.cc**

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

// Inserire qui le dichiarazioni

bool pal1 (double *sequenza, int primo, int ultimo);
bool palindroma (double *sequenza, int dim);

int main()
{
  int dim;
  double array[100];

  do
  {
    cout << "Inserisci il numero (0 < ... < 100) di elementi da memorizzare nell'array: ";
    cin >> dim;
  } while (dim <= 0);

  if (dim > 100)
  {
    cout << "Puoi inserire solo 100 elementi!" << endl;
    dim = 100;
  }
  for (int i = 0; i < dim; i++)
  {
    cout << "Inserire carattere " << i+1 <<" : ";
    cin >> array[i];
  }

  if ( palindroma(array, dim) )
    cout << "La sequenza inserita e' palindroma!" << endl;
  else
    cout << "La sequenza inserita NON e' palindroma!" << endl;

  return 0;
}

// Inserire qui le definizioni

bool pal1 (double *sequenza, int primo, int ultimo)
{
  if (primo > ultimo)
    return true;
  else if (sequenza[primo] != sequenza[ultimo])
    return false;
  else
    return pal1(sequenza, primo+1, ultimo-1);
}

bool palindroma (double *sequenza, int dim)
{
```

```
  return pal1(sequenza, 0, dim-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 `dim`-1 elementi;
- `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.

La coda deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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

3 esercizio3.cc

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

static int next(int index, int dim)
{
    return (index + 1) % dim;
}

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

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.dim) == 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.dim);
    return true;
}

bool dequeue(queue &q)
{
    if (is_empty(q)) {
        return false;
    }
    q.head = next(q.head, q.dim);
    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.dim)) {
        cout << q.elem[i] << " ";
    }
    cout << endl;
}
```

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

- `init` inizializzi la coda per contenere al più `dim`-1 elementi;
- `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.

La coda deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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

3 esercizio3.cc

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

static int next(int index, int dim)
{
    return (index + 1) % dim;
}

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

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.size) == q.head;
}

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

bool estrai_testa(queue &q)
{
    if (vuota(q)) {
        return false;
    }
    q.head = next(q.head, q.size);
    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.size)) {
        cout << q.elem[i] << " ";
    }
    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 `size`-1 elementi;
- `deinit` liberi la memoria utilizzata dalla coda;
- `print` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente.
- `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, la cui dimensione è specificata dall'argomento `size` della funzione `init`.

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

3 esercizio3.cc

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

static int next(int index, int dim)
{
    return (index + 1) % dim;
}

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

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.size) == 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.size);
    return true;
}

bool dequeue(queue &q)
{
    if (is_empty(q)) {
        return false;
    }
    q.head = next(q.head, q.size);
    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.size)) {
        cout << q.elem[i] << " ";
    }
    cout << endl;
}
```

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

- `init` inizializzi la coda per contenere al più `dim`-1 elementi;
- `deinit` liberi la memoria utilizzata dalla coda;
- `stampa` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente.
- `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, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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

3 esercizio3.cc

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

static int next(int index, int dim)
{
    return (index + 1) % dim;
}

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

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.dim) == q.head;
}

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

bool estrai_testa(queue &q)
{
    if (vuota(q)) {
        return false;
    }
    q.head = next(q.head, q.dim);
    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.dim)) {
        cout << q.elem[i] << " ";
    }
    cout << endl;
}
```

4 In un'espressione in Notazione Polacca (da non confondersi con la notazione polacca inversa vista a lezione) gli operatori si trovano tutti a sinistra degli argomenti. In tal modo, e' possibile rappresentare un'espressione complessa senza usare parentesi. Ad esempio:
"+ 3 4" è equivalente a "3+4",
"* + 3 4 - 6 3" è equivalente a "(3+4)*(6-3)",
"* + - 5 2 4 3" è equivalente a "((5-2)+4)*3".

Dato il file `esercizio4.cc` definire la funzione ricorsiva

```
double LeggiECalcolaEspressionePN();
```

che legge da tastiera un'espressione in Notazione Polacca e restituisca il valore risultante. Ad esempio:

```
PROMPT> ./a.out
> -3.4
-3.4
> + 3 4
7
> * + 3 4 - 6 3
21
> * + - 5 2 4 3
21
>
```

NOTA 1: assume che l'utente immetta sempre espressioni corrette.
NOTA 2: è possibile utilizzare la funzione `isdouble` definita nel programma.

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

4 **esercizio4.cc**

```
using namespace std;

#include <iostream>
#include <cstdio>


// restituisce true se e solo se la stringa s
// rappresenta un double, il cui valore viene
// restituito tramite la variabile x
bool isdouble(char * s,double & x) {
  return (sscanf(s,"%lf",&x) > 0);
}

// Definire qui sotto la funzione LeggiECalcolaEspressionePN()
// E' permesso definire all'occorrenza funzioni ausiliarie

double calcola (double op1, double op2, char op) {
  double res;
  switch (op) {
  case '+': res = op1+op2;break;
  case '-': res = op1-op2;break;
  case '*': res = op1*op2;break;
  case '/': res = op1/op2;break;
  }
  return res;
}



double LeggiECalcolaEspressionePN() {
  char s[100];
  double x,res,v1,v2;
  cin >> s;
  if (isdouble(s,x))
     res=x;
  else {// s contiene un operatore +,-,*./
    v1=LeggiECalcolaEspressionePN();
    v2=LeggiECalcolaEspressionePN();
    res = calcola (v1,v2,s[0]);
  }
  return res;
}


int main() {
double res;
  do {
    cout << "> ";
    res=LeggiECalcolaEspressionePN();
    cout << res << endl;
  } while (1);
}
```