Terzo Appello di Programmazione I

15 Giugno 2016
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 Secondo la formula di Taylor, la funzione seno iperbolico $sinh(x)$ può essere approssimata dalla successione

$$sinh_n(x) = \sum_{i=0}^{n} \frac{x^{2i+1}}{(2i+1)!} \tag{1}$$

$$\text{in modo tale che} \quad sinh(x) = \lim_{n \to +\infty} sinh_n(x) \tag{2}$$

Inoltre, se ϵ è l'errore massimo tollerabile, allora una possibile condizione di convergenza può essere espressa come:

$$|sinh_n(x) - sinh_{n-1}(x)| < \epsilon \cdot |sinh_{n-1}(x)| \tag{3}$$

Nel file `esercizio2.cc` è definito un programma per il calcolo approssimato della funzione **seno iperbolico**. Nello specifico, il programma richiede all'utente l'immissione di:

- un valore **intero** `nmax`, rappresentante il massimo numero n in (1)
- valore **double** `epsilon` (che rappresenta ϵ) in (3),
- una sequenza di numeri **double** `x` di cui calcolare il seno iperbolico approssimato.

Si richiede di completare il programma di cui sopra inserendo la dichiarazione e la definizione della funzione "`seno_iperbolico`", che, dati come parametri i valori `x`, `nmax` e `epsilon` di cui sopra, che calcoli $sinh_n(x)$ come in (1), terminando o quando la condizione (3) e' verificata oppure quando $n =$ `nmax`.

NOTE:

1. Non è consentito l'uso di funzioni di libreria, tranne che per l'I/O su console, né l'uso di variabili globali o di tipo `static`.
2. È ammessa, qualora ritenuta utile al completamento dell'esercizio, la definizione di funzioni ausiliarie (per esempio per il calcolo del fattoriale).
3. Non è consentito utilizzare istruzioni `break`, `continue` o `goto`.

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

1 **esercizio1.cc**

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

double valoreAssoluto(double valore);
double pot(double x, int n);
double fatt(double x);
double seno_iperbolico(double x, int nmax, double epsilon);

int main () {
  double epsilon, x;
  int nmax;

  cout << "Inserisci il massimo numero di termini dell'approssimazione nmax: ";
  cin >> nmax;
  cout << "Inserisci l'errore massimo epsilon: ";
  cin >> epsilon;

  char c;
  do {
    cout << "Inserisci il valore x in cui vuoi calcolare sinh(x): ";
    cin >> x;
    cout << "L'approssimazione di sinh(x) e': " << setprecision(10)
         << seno_iperbolico(x, nmax, epsilon) << endl;
    cout << "Continuare (s/n)? ";
    cin >> c;
  } while(c != 'n' && c != 'N');

  return(0);
}

double valoreAssoluto(double valore) {
  return valore < 0 ? -valore : valore;
}

double pot(double x, int n) {
  // Potenza (ricorsiva)
  if (n == 0) {
    return 1.0;
  } else {
    return x * pot(x, n - 1);
  }
}

double fatt(double x) {
  // Fattoriale (ricorsivo)
  if (x == 0.0) {
    return 1.0;
  } else {
    return x * fatt(x - 1);
  }
}
```

```
double seno_iperbolico(double x, int nmax, double epsilon) {
  double risultato, passo_precedente;
  // Condizioni iniziali
  risultato = 0.0;
  int i = 0;
  // Doppia condizione di uscita
  do {
    // Salvo lo step precedente
    passo_precedente = risultato;
    // Sommo l'i-esimo termine della serie
    risultato += pot(x, 2 * i + 1) / fatt(2 * i + 1);
    // Incremento il contatore
    i++;
  } while(i <= nmax &&
          valoreAssoluto(risultato - passo_precedente)
          >= epsilon * valoreAssoluto(passo_precedente));
  return risultato;
}
```

1 Secondo la formula di Taylor, la funzione coseno iperbolico $cosh(x)$ può essere approssimata dalla successione

$$cosh_n(x) = \sum_{i=0}^{n} \frac{x^{2i}}{(2i)!} \tag{4}$$

$$\text{in modo tale che} \quad cosh(x) = \lim_{n \to +\infty} cosh_n(x) \tag{5}$$

Inoltre, se ϵ è l'errore massimo tollerabile, allora una possibile condizione di convergenza può essere espressa come:

$$|cosh_n(x) - cosh_{n-1}(x)| < \epsilon \cdot |cosh_{n-1}(x)| \tag{6}$$

Nel file `esercizio2.cc` è definito un programma per il calcolo approssimato della funzione **coseno iperbolico**. Nello specifico, il programma richiede all'utente l'immissione di:

- un valore **intero** `nmax`, rappresentante il massimo numero n in (4)
- valore **double** `epsilon` (che rappresenta ϵ) in (6),
- una sequenza di numeri **double** `x` di cui calcolare il coseno iperbolico approssimato.

Si richiede di completare il programma di cui sopra inserendo la dichiarazione e la definizione della funzione "`coseno_iperbolico`", che, dati come parametri i valori `x`, `nmax` e `epsilon` di cui sopra, che calcoli $cosh_n(x)$ come in (4), terminando o quando la condizione (6) e' verificata oppure quando $n =$ `nmax`.

NOTE:

1. Non è consentito l'uso di funzioni di libreria, tranne che per l'I/O su console, né l'uso di variabili globali o di tipo `static`.
2. È ammessa, qualora ritenuta utile al completamento dell'esercizio, la definizione di funzioni ausiliarie (per esempio per il calcolo del fattoriale).
3. Non è consentito utilizzare istruzioni `break`, `continue` o `goto`.

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

1 **esercizio1.cc**

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

double valoreAssoluto(double valore);
double pot(double x, int n);
double fatt(double x);
double coseno_iperbolico(double x, int nmax, double epsilon);

int main () {
  double epsilon, x;
  int nmax;

  cout << "Inserisci il massimo numero di termini dell'approssimazione nmax: ";
  cin >> nmax;
  cout << "Inserisci l'errore massimo epsilon: ";
  cin >> epsilon;

  char c;
  do {
    cout << "Inserisci il valore x in cui vuoi calcolare cosh(x): ";
    cin >> x;
    cout << "L'approssimazione di cosh(x) e': " << setprecision(10)
         << coseno_iperbolico(x, nmax, epsilon) << endl;
    cout << "Continuare (s/n)? ";
    cin >> c;
  } while(c != 'n' && c != 'N');

  return(0);
}

double valoreAssoluto(double valore) {
  return valore < 0 ? -valore : valore;
}

double pot(double x, int n) {
  // Potenza (ricorsiva)
  if (n == 0) {
    return 1.0;
  } else {
    return x * pot(x, n - 1);
  }
}

double fatt(double x) {
  // Fattoriale (ricorsivo)
  if (x == 0.0) {
    return 1.0;
  } else {
    return x * fatt(x - 1);
  }
}
```

```
double coseno_iperbolico(double x, int nmax, double epsilon) {
  double risultato, passo_precedente;
  // Condizioni iniziali
  risultato = 0.0;
  int i = 0;
  // Doppia condizione di uscita
  do {
    // Salvo lo step precedente
    passo_precedente = risultato;
    // Sommo l'i-esimo termine della serie
    risultato += pot(x, 2 * i) / fatt(2 * i);
    // Incremento il contatore
    i++;
  } while(i <= nmax &&
          valoreAssoluto(risultato - passo_precedente)
          >= epsilon * valoreAssoluto(passo_precedente));
  return risultato;
}
```

1 Secondo la formula di Taylor, la funzione "meno seno iperbolico" $-sinh(x)$ può essere approssimata dalla successione

$$-sinh_n(x) = \sum_{i=0}^{n} -\frac{x^{2i+1}}{(2i+1)!} \tag{7}$$

in modo tale che $$-sinh(x) = \lim_{n \to +\infty} -sinh_n(x) \tag{8}$$

Inoltre, se ϵ è l'errore massimo tollerabile, allora una possibile condizione di convergenza può essere espressa come:

$$| - sinh_n(x) - -sinh_{n-1}(x)| < \epsilon \cdot | - sinh_{n-1}(x)| \tag{9}$$

Nel file `esercizio2.cc` è definito un programma per il calcolo approssimato della funzione **meno seno iperbolico**. Nello specifico, il programma richiede all'utente l'immissione di:

- un valore **intero** `nmax`, rappresentante il massimo numero n in (7)
- valore **double** `epsilon` (che rappresenta ϵ) in (9),
- una sequenza di numeri **double** `x` di cui calcolare l'opposto del seno iperbolico approssimato.

Si richiede di completare il programma di cui sopra inserendo la dichiarazione e la definizione della funzione "`meno_seno_iperbolico`", che, dati come parametri i valori `x`, `nmax` e `epsilon` di cui sopra, che calcoli $-sinh_n(x)$ come in (7), terminando o quando la condizione (9) e' verificata oppure quando $n =$ `nmax`.

NOTE:

1. Non è consentito l'uso di funzioni di libreria, tranne che per l'I/O su console, né l'uso di variabili globali o di tipo `static`.
2. È ammessa, qualora ritenuta utile al completamento dell'esercizio, la definizione di funzioni ausiliarie (per esempio per il calcolo del fattoriale).
3. Non è consentito utilizzare istruzioni `break`, `continue` o `goto`.

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

1 **esercizio1.cc**

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

double valoreAssoluto(double valore);
double pot(double x, int n);
double fatt(double x);
double meno_seno_iperbolico(double x, int nmax, double epsilon);

int main () {
  double epsilon, x;
  int nmax;

  cout << "Inserisci il massimo numero di termini dell'approssimazione nmax: ";
  cin >> nmax;
  cout << "Inserisci l'errore massimo epsilon: ";
  cin >> epsilon;

  char c;
  do {
    cout << "Inserisci il valore x in cui vuoi calcolare -sinh(x): ";
    cin >> x;
    cout << "L'approssimazione di -sinh(x) e': " << setprecision(10)
         << meno_seno_iperbolico(x, nmax, epsilon) << endl;
    cout << "Continuare (s/n)? ";
    cin >> c;
  } while(c != 'n' && c != 'N');

  return(0);
}

double valoreAssoluto(double valore) {
  return valore < 0 ? -valore : valore;
}

double pot(double x, int n) {
  // Potenza (ricorsiva)
  if (n == 0) {
    return 1.0;
  } else {
    return x * pot(x, n - 1);
  }
}

double fatt(double x) {
  // Fattoriale (ricorsivo)
  if (x == 0.0) {
    return 1.0;
  } else {
    return x * fatt(x - 1);
  }
```

```
}

double meno_seno_iperbolico(double x, int nmax, double epsilon) {
  double risultato, passo_precedente;
  // Condizioni iniziali
  risultato = 0.0;
  int i = 0;
  // Doppia condizione di uscita
  do {
    // Salvo lo step precedente
    passo_precedente = risultato;
    // Sottraggo l'i-esimo termine della serie
    risultato -= pot(x, 2 * i + 1) / fatt(2 * i + 1);
    // Incremento il contatore
    i++;
  } while(i <= nmax &&
          valoreAssoluto(risultato - passo_precedente)
          >= epsilon * valoreAssoluto(passo_precedente));
  return risultato;
}
```

1 Secondo la formula di Taylor, la funzione "meno coseno iperbolico" $-cosh(x)$ può essere approssimata dalla successione

$$-cosh_n(x) = \sum_{i=0}^{n} -\frac{x^{2i}}{(2i)!} \tag{10}$$

in modo tale che $$-cosh(x) = \lim_{n \to +\infty} -cosh_n(x) \tag{11}$$

Inoltre, se ϵ è l'errore massimo tollerabile, allora una possibile condizione di convergenza può essere espressa come:

$$| - cosh_n(x) - -cosh_{n-1}(x)| < \epsilon \cdot | - cosh_{n-1}(x)| \tag{12}$$

Nel file `esercizio2.cc` è definito un programma per il calcolo approssimato della funzione **coseno iperbolico**. Nello specifico, il programma richiede all'utente l'immissione di:

- un valore **intero** `nmax`, rappresentante il massimo numero n in (10)
- valore **double** `epsilon` (che rappresenta ϵ) in (12),
- una sequenza di numeri **double** `x` di cui calcolare l'opposto del coseno iperbolico approssimato.

Si richiede di completare il programma di cui sopra inserendo la dichiarazione e la definizione della funzione "`meno_coseno_iperbolico`", che, dati come parametri i valori `x`, `nmax` e `epsilon` di cui sopra, che calcoli $-cosh_n(x)$ come in (10), terminando o quando la condizione (12) e' verificata oppure quando $n =$ `nmax`.

NOTE:

1. Non è consentito l'uso di funzioni di libreria, tranne che per l'I/O su console, né l'uso di variabili globali o di tipo `static`.
2. È ammessa, qualora ritenuta utile al completamento dell'esercizio, la definizione di funzioni ausiliarie (per esempio per il calcolo del fattoriale).
3. Non è consentito utilizzare istruzioni `break`, `continue` o `goto`.

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

1 **esercizio1.cc**

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

double valoreAssoluto(double valore);
double pot(double x, int n);
double fatt(double x);
double meno_coseno_iperbolico(double x, int nmax, double epsilon);

int main () {
  double epsilon, x;
  int nmax;

  cout << "Inserisci il massimo numero di termini dell'approssimazione nmax: ";
  cin >> nmax;
  cout << "Inserisci l'errore massimo epsilon: ";
  cin >> epsilon;

  char c;
  do {
    cout << "Inserisci il valore x in cui vuoi calcolare cosh(x): ";
    cin >> x;
    cout << "L'approssimazione di -cosh(x) e': " << setprecision(10)
         << meno_coseno_iperbolico(x, nmax, epsilon) << endl;
    cout << "Continuare (s/n)? ";
    cin >> c;
  } while(c != 'n' && c != 'N');

  return(0);
}

double valoreAssoluto(double valore) {
  return valore < 0 ? -valore : valore;
}

double pot(double x, int n) {
  // Potenza (ricorsiva)
  if (n == 0) {
    return 1.0;
  } else {
    return x * pot(x, n - 1);
  }
}

double fatt(double x) {
  // Fattoriale (ricorsivo)
  if (x == 0.0) {
    return 1.0;
  } else {
    return x * fatt(x - 1);
  }
}
```

```
double meno_coseno_iperbolico(double x, int nmax, double epsilon) {
  double risultato, passo_precedente;
  // Condizioni iniziali
  risultato = 0.0;
  int i = 0;
  // Doppia condizione di uscita
  do {
    // Salvo lo step precedente
    passo_precedente = risultato;
    // Sommo l'i-esimo termine della serie
    risultato -= pot(x, 2 * i) / fatt(2 * i);
    // Incremento il contatore
    i++;
  } while(i <= nmax &&
          valoreAssoluto(risultato - passo_precedente)
          >= epsilon * valoreAssoluto(passo_precedente));
  return risultato;
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `invert` che inverta l'array di caratteri passato come parametro. Per esempio, dato l'array $\{a, b, c, d, e\}$, l'ordine dei caratteri deve essere invertito ottenendo $\{e, d, c, b, a\}$.

NOTA 1: **La funzione `invert` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Si può fare uso di eventuali funzioni ricorsive ausiliarie.**

NOTA 2: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

// Dichiarare qui sotto la funzione invert
void invert(char in[], int dim);
void invert_ric(char in[], int i, int l);

int main () {
   char sequence[30];
   int dim;

   cout << "Dimensione: ";
   cin >> dim;
   cout << "Array: ";
   for (int i = 0; i < dim; i++) {
      cin >> sequence[i];
   }

   invert(sequence, dim);

   cout << "Array invertito: ";
   for (int i = 0; i < dim; i++) {
      cout << sequence[i] << " ";
   }
   cout << endl;

   return 0;
}


// Definire qui sotto la funzione invert

void invert(char in[], int dim) {
   invert_ric(in,0,dim);
}

void invert_ric(char in[], int i, int dim) {
   if (i >= dim/2) {
      return;
   } else {
      int tmp = in[dim-1-i];
      in[dim-1-i] = in[i];
      in[i] = tmp;
      invert_ric(in, i+1, dim);
   }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `inverti_array` che inverta l'array di interi passato come parametro. Per esempio, dato l'array $\{1, 2, 3, 4, 5\}$, l'ordine dei caratteri deve essere invertito ottenendo $\{5, 4, 3, 2, 1\}$.

NOTA 1: **La funzione `inverti_array` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Si può fare uso di eventuali funzioni ricorsive ausiliarie.**

NOTA 2: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

// Dichiarare qui sotto la funzione inverti_array
void inverti_array(int in[], int dim);
void inverti_array_ric(int in[], int i, int l);

int main () {
   int array[30], dim;

   cout << "Dimensione: ";
   cin >> dim;
   cout << "Array: ";
   for (int i = 0; i < dim; i++) {
      cin >> array[i];
   }

   inverti_array(array, dim);

   cout << "Array invertito: ";
   for (int i = 0; i < dim; i++) {
      cout << array[i] << " ";
   }
   cout << endl;

   return 0;
}


// Definire qui sotto la funzione inverti_array

void inverti_array(int in[], int dim) {
   inverti_array_ric(in,0,dim);
}

void inverti_array_ric(int in[], int i, int dim) {
   if (i >= dim/2) {
      return;
   } else {
      int tmp = in[dim-1-i];
      in[dim-1-i] = in[i];
      in[i] = tmp;
      inverti_array_ric(in, i+1, dim);
   }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `reverse` che inverta l'array di `long` passato come parametro. Per esempio, dato l'array $\{4, 9, 8, 7, 5\}$, l'ordine dei numeri deve essere invertito ottenendo $\{5, 7, 8, 9, 4\}$.

NOTA 1: **La funzione `reverse` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Si può fare uso di eventuali funzioni ricorsive ausiliarie.**

NOTA 2: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

// Dichiarare qui sotto la funzione reverse
void reverse(long in[], int dim);
void reverse_ric(long in[], int i, int l);

int main () {
   long sequence[30];
   int dim;

   cout << "Dimensione: ";
   cin >> dim;
   cout << "Array: ";
   for (int i = 0; i < dim; i++) {
      cin >> sequence[i];
   }

   reverse(sequence, dim);

   cout << "Array invertito: ";
   for (int i = 0; i < dim; i++) {
      cout << sequence[i] << " ";
   }
   cout << endl;

   return 0;
}


// Definire qui sotto la funzione reverse

void reverse(long in[], int dim) {
   reverse_ric(in,0,dim);
}

void reverse_ric(long in[], int i, int dim) {
   if (i >= dim/2) {
      return;
   } else {
      int tmp = in[dim-1-i];
      in[dim-1-i] = in[i];
      in[i] = tmp;
      reverse_ric(in, i+1, dim);
   }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `rovescia_array` che inverta l'array di `float` passato come parametro. Per esempio, dato l'array $\{1.5, 2.1, 3, 4.4, 5\}$, l'ordine dei caratteri deve essere invertito ottenendo $\{5, 4.4, 3, 2.1, 1.5\}$.

NOTA 1: **La funzione `rovescia_array` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Si può fare uso di eventuali funzioni ricorsive ausiliarie.**

NOTA 2: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

// Dichiarare qui sotto la funzione rovescia_array
void rovescia_array(float in[], int dim);
void rovescia_array_ric(float in[], int i, int l);

int main () {
   float array[30], dim;

   cout << "Dimensione: ";
   cin >> dim;
   cout << "Array: ";
   for (int i = 0; i < dim; i++) {
      cin >> array[i];
   }

   rovescia_array(array, dim);

   cout << "Array rovesciato: ";
   for (int i = 0; i < dim; i++) {
      cout << array[i] << " ";
   }
   cout << endl;

   return 0;
}


// Definire qui sotto la funzione rovescia_array

void rovescia_array(float in[], int dim) {
   rovescia_array_ric(in,0,dim);
}

void rovescia_array_ric(float in[], int i, int dim) {
   if (i >= dim/2) {
      return;
   } else {
      int tmp = in[dim-1-i];
      in[dim-1-i] = in[i];
      in[i] = tmp;
      rovescia_array_ric(in, i+1, dim);
   }
}
```

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_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.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 soluzione_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_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.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 soluzione_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_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.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
```

3 soluzione_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_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.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 soluzione_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;
}
```

4 Scrivere nel file **esercizio4.cc** un programma, contenente esclusivamente funzioni **ricorsive**, che prenda come argomento una singola parola caratterizzata da una sequenza (anche vuota) di '+' e '-' seguita da una sequenza non vuota di cifre, e stampi:

- il numero corrispondente alla sequanza di cifre, **positivo** se la sequenza contiene un numero pari di '-', **negativo** se la sequenza contiene un numero dispari di '-';
- il numero suddetto moltiplicato per 5.

secondo il formato degli esempi qui sotto.

ESEMPI:

```
~> ./a.out 4
Ho letto 4, il numero moltiplicato per 5 e': 20
~> ./a.out -4
Ho letto -4, il numero moltiplicato per 5 e': -20
~> ./a.out -+-+-4
Ho letto -4, il numero moltiplicato per 5 e': -20
~> ./a.out -++-4
Ho letto 4, il numero moltiplicato per 5 e': 20
~> ./a.out -++-346
Ho letto 346, il numero moltiplicato per 5 e': 1730
~> ./a.out -++-+-+--346
Ho letto -346, il numero moltiplicato per 5 e': -1730
~> ./a.out -++-+-+--046
Ho letto -46, il numero moltiplicato per 5 e': -230
~> ./a.out -++-+-+-046
Ho letto 46, il numero moltiplicato per 5 e': 230
~>
```

NOTA 1: non sono ammessi cicli, né invocazioni di funzioni che non siano ricorsive.

NOTA 2: è vietato altresì l'uso di istruzioni `break`, `continue`, `goto` e di ogni altra forma di salto non condizionato.

NOTA 3: è ammesso invece l'uso di funzioni wrapper, se ritenute utili.

NOTA 4: non è ammesso l'uso di funzioni di libreria, tranne che per l'I/O su console.

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

4 **esercizio4.cc**

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

int converti (char * s,int sign,int currvalue) {
  int res;
  if (*(s+1)=='\0')
    res = sign*(currvalue*10+(*s-'0'));
  else if ((*s>='0')&&(*s<='9'))  // *s is a digit
    res = converti(s+1,sign,currvalue*10+(*s-'0'));
  else if (*s == '+') // *s is '-'
    res = converti(s+1,sign,currvalue);
  else if (*s == '-') // *s is '-'
    res = converti(s+1,sign*-1,currvalue);
  return res;
}

int main (int argc, char * argv[]) {
  cout << "Ho letto " << converti(argv[1],1,0) <<
          ", il numero moltiplicato per 5 e': " <<
          5*converti(argv[1],1,0) << endl;
}
```