# Secondo Appello di Programmazione I

13 Febbraio 2013
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 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main`:

- il nome di un file di testo contenente una lista non ordinata di voti compresi tra 0 e 30,
- un intero `DIM` che rappresenta il numero di voti presenti nel file di testo,
- un intero `N` minore o uguale a `DIM`,

crei un nuovo file di testo (chiamato `output.txt`) contenente una lista ordinata con gli `N` voti più alti presenti nel file passato come argomento da linea di comando.

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

```
17
29
25
30
15
27
23
18
```

allora il comando:

```
./a.out input.txt 8 4
```

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

```
30
29
27
25
```

NOTA 1: Utilizzare la funzione `sort_array()` fornita nei file `sorting.h` e `sorting.o`.

NOTA 2: Per convertire una stringa in intero è ammesso l'uso della funzione `atoi()` della libreria `<cstdlib>`.

NOTA 3: Il programma non deve prevedere alcun numero massimo di voti leggibili dal file, pena l'annullamento dell'esercizio.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include "sorting.h"

using namespace std;

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

  fstream my_in, my_out;
  int DIM, N, voto;

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile> <dim> <n>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open("output.txt",ios::out);

  DIM = atoi(argv[2]);
  N = atoi(argv[3]);

  int* array_voti = new int[DIM];

  int num_voti = 0;
  while(my_in >> voto){
     array_voti[num_voti++] = voto;
  }

  sort_array(array_voti, DIM);

  for(int i=0; i<N; i++) {
     my_out << array_voti[i] << "\n";
  }

  delete[] array_voti;

  my_in.close();
  my_out.close();

  return 0;
}
```

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

- il nome di un file di testo contenente una lista non ordinata di caratteri,
- un intero `DIM` che rappresenta il numero di caratteri presenti nel file di testo,
- un intero `N` minore o uguale a `DIM`,

crei un nuovo file di testo (chiamato `output.txt`) contenente una lista ordinata con gli `N` caratti più bassi presenti nel file passato come argomento da linea di comando. Si assuma che il carattere $z$ sia più basso di $y$, $y$ sia più basso di $x$, $x$ sia più basso di $w$, etc.

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

```
s
a
n
e
p
c
d
h
y
```

allora il comando:

```
./a.out input.txt 9 4
```

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

```
y
s
p
n
```

NOTA 1: Utilizzare la funzione `ordina_array()` fornita nei file `ordinamento.h` e `ordinamento.o`.

NOTA 2: Per convertire una stringa in intero è ammesso l'uso della funzione `atoi()` della libreria `<cstdlib>`.

NOTA 3: Il programma non deve prevedere alcun numero massimo di caratteri leggibili dal file, pena l'annullamento dell'esercizio.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include "ordinamento.h"

using namespace std;

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

  fstream my_in, my_out;
  int DIM, N;
  char carattere;

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile> <dim> <n>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open("output.txt",ios::out);

  DIM = atoi(argv[2]);
  N = atoi(argv[3]);

  char* array_carattere = new char[DIM];

  int num_caratteri = 0;
  while(my_in >> carattere){
     array_carattere[num_caratteri++] = carattere;
  }

  ordina_array(array_carattere, DIM);

  for(int i=DIM-1; i>=DIM-N; i--) {
     my_out << array_carattere[i] << "\n";
  }

  delete[] array_carattere;

  my_in.close();
  my_out.close();

  return 0;
}
```

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

- il nome di un file di testo contenente una lista non ordinata di voti compresi tra 0 e 30,
- un intero `DIM` che rappresenta il numero di voti presenti nel file di testo,
- un intero `N` minore o uguale a `DIM`,

crei un nuovo file di testo (chiamato `output.txt`) contenente una lista ordinata con gli `N` voti più bassi presenti nel file passato come argomento da linea di comando.

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

```
17
28
24
30
20
19
15
27
22
```

allora il comando:

```
./a.out input.txt 9 4
```

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

```
15
17
19
20
```

NOTA 1: Utilizzare la funzione `sort_array()` fornita nei file `sorting.h` e `sorting.o`.

NOTA 2: Per convertire una stringa in intero è ammesso l'uso della funzione `atoi()` della libreria `<cstdlib>`.

NOTA 3: Il programma non deve prevedere alcun numero massimo di voti leggibili dal file, pena l'annullamento dell'esercizio.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include "sorting.h"

using namespace std;

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

  fstream my_in, my_out;
  int DIM, N, voto;

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile> <dim> <n>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open("output.txt",ios::out);

  DIM = atoi(argv[2]);
  N = atoi(argv[3]);

  int* array_voti = new int[DIM];

  int num_voti = 0;
  while(my_in >> voto){
     array_voti[num_voti++] = voto;
  }

  sort_array(array_voti, DIM);

  for(int i=DIM-1; i>=DIM-N; i--) {
     my_out << array_voti[i] << "\n";
  }

  delete[] array_voti;

  my_in.close();
  my_out.close();

  return 0;
}
```

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

- il nome di un file di testo contenente una lista non ordinata di caratteri,
- un intero `DIM` che rappresenta il numero di caratteri presenti nel file di testo,
- un intero `N` minore o uguale a `DIM`,

crei un nuovo file di testo (chiamato `output.txt`) contenente una lista ordinata con gli `N` caratti più alti presenti nel file passato come argomento da linea di comando. Si assuma che il carattere $a$ sia più alto di $b$, $b$ sia più alto di $c$, $c$ sia più alto di $d$, etc.

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

```
g
s
m
a
f
p
c
d
i
x
```

allora il comando:

```
./a.out input.txt 10 3
```

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

```
a
c
d
```

NOTA 1: Utilizzare la funzione `ordina_array()` fornita nei file `ordinamento.h` e `ordinamento.o`.

NOTA 2: Per convertire una stringa in intero è ammesso l'uso della funzione `atoi()` della libreria `<cstdlib>`.

NOTA 3: Il programma non deve prevedere alcun numero massimo di caratteri leggibili dal file, pena l'annullamento dell'esercizio.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include "ordinamento.h"

using namespace std;

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

  fstream my_in, my_out;
  int DIM, N;
  char carattere;

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile> <dim> <n>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open("output.txt",ios::out);

  DIM = atoi(argv[2]);
  N = atoi(argv[3]);

  char* array_carattere = new char[DIM];

  int num_caratteri = 0;
  while(my_in >> carattere){
     array_carattere[num_caratteri++] = carattere;
  }

  ordina_array(array_carattere, DIM);

  for(int i=0; i<N; i++) {
     my_out << array_carattere[i] << "\n";
  }

  delete[] array_carattere;

  my_in.close();
  my_out.close();

  return 0;
}
```

2 Secondo la formula di Taylor, la funzione seno $sen(x)$ può essere descritta in forma di serie come segue:

$$sen(x) = \sum_{i=0}^{+\infty} \frac{(-1)^i}{(2i+1)!} x^{2i+1}$$

Scrivere nel file `esercizio2.cc`:

- la dichiarazione e la definizione della **funzione ricorsiva** `fatt` che, data una variable di tipo `double` $x$, ritorni il fattoriale di $x$ in forma di `double` (si assuma $x$ positiva);
- la dichiarazione e la definizione della **funzione ricorsiva** `pot` che, data una variable di tipo `double` $x$ ed una variabile intera $n$, ritorni la potenza $x^n$ in forma di `double` (si assuma $n$ positiva);
- la dichiarazione e la definizione della **funzione ricorsiva** `my_sen`, che, dati come parametri una variabile di tipo `double` $x$ ed una variabile intera $n$, restituisca un'approssimazione della funzione $sen(x)$ che considera un numero finito $N+1$ di termini della serie di cui sopra.

NOTA 1: **Le funzioni `fatt`, `pot`, `my_sen` devono essere ricorsive ed al loro interno non ci possono essere cicli (o "goto") o chiamate a funzioni contenenti cicli (o "goto").**

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>
#include <cmath>
using namespace std;

// Inserire qui sotto la dichiarazione delle funzioni
double pot(double x, int n);
double fatt(double x);
double my_sin(double x, int n);

int main () {
  double x, eps;
  int n;

  cout << "Inserisci il valore x in cui vuoi calcolare sin(x): \n";
  cin >> x;
  cout << "Inserisci il numero di termini dell'approssimazione: \n";
  cin >> n;
  cout << "L'approssimazione di sin(x) e': " << my_sin(x, n) << endl;
  cout << "Per confronto, il valore di sin(x) e': " << sin(x) << endl;
  return(0);
}

// Inserire qui sotto la definizione delle funzioni

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

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

double my_sin(double x, int n) {
   if (n < 0) {
      return 0.0;
   } else {
      return (pot(-1,n)/fatt(2*n+1))*pot(x,2*n+1)+my_sin(x,n-1);
   }
}
```

2 Secondo la formula di Taylor, la funzione coseno iperbolico $cosh(x)$ può essere descritta in forma di serie come segue:

$$cosh(x) = \sum_{i=0}^{+\infty} \frac{x^{2i}}{(2i)!}$$

Scrivere nel file `esercizio2.cc`:

- la dichiarazione e la definizione della **funzione ricorsiva** `fatt` che, data una variable di tipo `double` $x$, ritorni il fattoriale di $x$ in forma di `double` (si assuma $x$ positiva);
- la dichiarazione e la definizione della **funzione ricorsiva** `pot` che, data una variable di tipo `double` $x$ ed una variabile intera $n$, ritorni la potenza $x^n$ in forma di `double` (si assuma $n$ positiva);
- la dichiarazione e la definizione della **funzione ricorsiva** `my_cosh`, che, dati come parametri una variabile di tipo `double` $x$ ed una variabile intera $n$, restituisca un'approssimazione della funzione $cosh(x)$ che considera un numero finito $N+1$ di termini della serie di cui sopra.

NOTA 1: **Le funzioni `fatt`, `pot`, `my_cosh` devono essere ricorsive ed al loro interno non ci possono essere cicli (o "goto") o chiamate a funzioni contenenti cicli (o "goto").**

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>
#include <cmath>
using namespace std;

// Inserire qui sotto la dichiarazione delle funzioni
double pot(double x, int n);
double fatt(double x);
double my_cosh(double x, int n);

int main () {
  double x, eps;
  int n;

  cout << "Inserisci il valore x in cui vuoi calcolare cosh(x): \n";
  cin >> x;
  cout << "Inserisci il numero di termini dell'approssimazione: \n";
  cin >> n;
  cout << "L'approssimazione di cosh(x) e': " << my_cosh(x, n) << endl;
  cout << "Per confronto, il valore di cosh(x) e': " << cosh(x) << endl;
  return(0);
}

// Inserire qui sotto la definizione delle funzioni

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

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

double my_cosh(double x, int n) {
   if (n < 0) {
      return 0.0;
   } else {
      return (pot(x,2*n)/fatt(2*n))+my_cosh(x,n-1);
   }
}
```

2 Secondo la formula di Taylor, la funzione seno iperbolico $sinh(x)$ può essere descritta in forma di serie come segue:

$$sinh(x) = \sum_{i=0}^{+\infty} \frac{x^{2i+1}}{(2i+1)!}.$$

Scrivere nel file `esercizio2.cc`:

- la dichiarazione e la definizione della **funzione ricorsiva** `fatt` che, data una variable di tipo `double` $x$, ritorni il fattoriale di $x$ in forma di `double` (si assuma $x$ positiva);
- la dichiarazione e la definizione della **funzione ricorsiva** `pot` che, data una variable di tipo `double` $x$ ed una variabile intera $n$, ritorni la potenza $x^n$ in forma di `double` (si assuma $n$ positiva);
- la dichiarazione e la definizione della **funzione ricorsiva** `my_sinh` che, dati come parametri una variabile di tipo `double` $x$ ed una variabile intera $n$, restituisca un'approssimazione della funzione $sinh(x)$ che considera un numero finito $N+1$ di termini della serie di cui sopra.

NOTA 1: **Le funzioni `fatt`, `pot`, `my_sinh` devono essere ricorsive ed al loro interno non ci possono essere cicli (o "goto") o chiamate a funzioni contenenti cicli (o "goto").**

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>
#include <cmath>
using namespace std;

// Inserire qui sotto la dichiarazione delle funzioni
double pot(double x, int n);
double fatt(double x);
double my_sinh(double x, int n);

int main () {
  double x, eps;
  int n;

  cout << "Inserisci il valore x in cui vuoi calcolare sinh(x): \n";
  cin >> x;
  cout << "Inserisci il numero di termini dell'approssimazione: \n";
  cin >> n;
  cout << "L'approssimazione di sinh(x) e': " << my_sinh(x, n) << endl;
  cout << "Per confronto, il valore di sinh(x) e': " << sinh(x) << endl;
  return(0);
}

// Inserire qui sotto la definizione delle funzioni

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

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

double my_sinh(double x, int n) {
   if (n < 0) {
      return 0.0;
   } else {
      return (pot(x,2*n+1)/fatt(2*n+1))+my_sinh(x,n-1);
   }
}
```

2 Secondo la formula di Taylor, la funzione coseno $cos(x)$ può essere descritta in forma di serie come segue:

$$cos(x) = \sum_{i=0}^{+\infty} \frac{(-1)^i}{(2i)!} x^{2i}$$

Scrivere nel file `esercizio2.cc`:

- la dichiarazione e la definizione della **funzione ricorsiva** `fatt` che, data una variable di tipo `double` $x$, ritorni il fattoriale di $x$ in forma di `double` (si assuma $x$ positiva);
- la dichiarazione e la definizione della **funzione ricorsiva** `pot` che, data una variable di tipo `double` $x$ ed una variabile intera $n$, ritorni la potenza $x^n$ in forma di `double` (si assuma $n$ positiva);
- la dichiarazione e la definizione della **funzione ricorsiva** `my_cos`, che, dati come parametri una variabile di tipo `double` $x$ ed una variabile intera $n$, restituisca un'approssimazione della funzione $cos(x)$ che considera un numero finito $N + 1$ di termini della serie di cui sopra.

NOTA 1: **Le funzioni `fatt`, `pot`, `my_cos` devono essere ricorsive ed al loro interno non ci possono essere cicli (o "goto") o chiamate a funzioni contenenti cicli (o "goto").**

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>
#include <cmath>
using namespace std;

// Inserire qui sotto la dichiarazione delle funzioni
double pot(double x, int n);
double fatt(double x);
double my_cos(double x, int n);

int main () {
  double x, eps;
  int n;

  cout << "Inserisci il valore x in cui vuoi calcolare cos(x): \n";
  cin >> x;
  cout << "Inserisci il numero di termini dell'approssimazione: \n";
  cin >> n;
  cout << "L'approssimazione di cos(x) e': " << my_cos(x, n) << endl;
  cout << "Per confronto, il valore di cos(x) e': " << cos(x) << endl;
  return(0);
}

// Inserire qui sotto la definizione delle funzioni

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

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

double my_cos(double x, int n) {
   if (n < 0) {
      return 0.0;
   } else {
      return (pot(-1,n)/fatt(2*n))*pot(x,2*n)+my_cos(x,n-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, e il numero massimo di elementi che possono essere inseriti nella coda e' specificato dall'argomento `maxnum` della funzione `init`.

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

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

3 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 `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, e il numero massimo di elementi che possono essere inseriti nella coda e' specificato dall'argomento `maxnum` della funzione `init`.

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

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

3 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, e il numero massimo di elementi che possono essere inseriti nella coda e' specificato dall'argomento `maxnum` della funzione `init`.

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

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

3 **albero_main.cc**

3 **albero.h**

3 **soluzione_A33.cc**

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

- `init` inizializzi la coda per contenere al più `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, 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 6 punti (al punteggio di tutti gli esercizi va poi sommato 10).

4 Si definisca una funzione **ricorsiva** "`potenza`" che prenda in ingresso due numeri interi positivi $n$ e $m$ e restituisca un valore intero che rappresenti l'operazione di elevamento a potenza $n^m$, utilizzando esclusivamente gli operatori di confronto `==,!=,>=,>,<=,<` e le funzioni predefinite `succ` e `pred`, che calcolano il successore e il predecessore di un numero intero.

Alcune note:

(a) non è consentito utilizzare gli operatori aritmetici `+,-,*, /, %`,

(b) non è consentito utilizzare alcuna funzione di libreria;

(c) non è consentito utilizzare alcuna forma di ciclo;

(d) è consentito definire e utilizzare eventuali funzioni ausiliarie, purché a loro volta rispettino le condizioni (a), (b) e (c).

Sono dati il seguente file header `predsucc.h`:

```
// se n>=0 restituisce n+1, altrimenti abortisce il programma
int succ(int n);
// se n>=1 restituisce n-1, altrimenti abortisce il programma
int pred(int n);
```

e il corrispondente file `predsucc.o` dove sono definite `succ` e `pred`, e il seguente file principale:

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

// introdurre qui la definizione della funzione potenza

int main()
{
  int n,m;

  do {
    cout << "dammi due numeri interi positivi: ";
    cin >> n >> m;
    if (n < 0 || m < 0)
      cout << "Valori errati, programma terminato." << endl;
    else
      cout << n << "^" << m << " = " << potenza(n,m) << endl;
  }
  while (n >= 0 && m >= 0);
}
```

dove deve essere definita la funzione `prodotto`.

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