Quarto Appello di Programmazione I

08 Luglio 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 esame

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

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

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

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

1 Mediante la formula di Taylor si può ottenere un'approsimazione del valore della funzione $sin(x)$ dove x è espresso in radianti. In particolare con:

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

è possibile avere un'approssimazione della funzione seno considerando un certo numero di termini di questa sommatoria. Data una certa tolleranza ε ("epsilon") è possibile valutare la bontà di un'approssimazione tramite la seguente formula:

$$\left| \frac{f_n(x) - f_{n-1}(x)}{f_n(x)} \right| < \varepsilon,$$

dove $f_n(x)$ è la sommatoria descritta precedentemente.

Nel file `esercizio1.cc` scrivere la definizione della funzione `approssimazione_sin` in grado di calcolare e ritornare, usando la sommatoria sopra descritta, un'approssimazione della funzione seno, calcolato in `x`. Tale funzione deve terminare quando l'approssimazione calcolata rientra in una data tolleranza `eps`, oppure dopo un certo numero `max_iter` di passi.

Si utilizzino le seguenti funzioni della libreria `math.h`:

```
double fabs (double x);
double pow (double x, double y);
```

che restituiscono rispettivamente il valore assoluto di un numero reale `x` e il valore di `x` elevato alla `y`, e la funzione `fattoriale` data.

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

1 **esercizio1.cc**

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

double fattoriale (double x);
double approssimazione_sin (double x, double eps, int max_iter);

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

  cout << "Inserisci il valore x, espresso in radianti, in cui vuoi calcolare sin(x): \n";
  cin >> x;
  cout << "Inserisci la tolleranza eps: \n";
  cin >> eps;
  cout << "Inserisci il numero massimo di passi da eseguire: \n";
  cin >> n;

  cout << "L'approssimazione di sin(x) e': "
       << approssimazione_sin (x, eps, n) << endl;

  return(0);
}

double fattoriale (double x)
{
  double valore=1.0, n;

  for (n=x; n>0; n--)
    {
      valore=valore*n;
    }

  return(valore);
}

double approssimazione_sin (double x, double eps, int n) {

  double app = 0, controllo;
  int i=0;

  do {
    double old;

    old = app;
    app = app + (pow ((double) -1,i) * pow (x,(2*i)+1) / fattoriale((2*i)+1));
    controllo = fabs((app - old) / app);
    i++;

  } while ((controllo >= eps)&&(i<n));
```

```
  return (app);
}
```

1 Mediante la formula di Taylor si può ottenere un'approsimazione del valore della funzione $cos(x)$ dove x è espresso in radianti. In particolare con:

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

è possibile avere un'approssimazione della funzione coseno considerando un certo numero di termini di questa sommatoria. Data una certa tolleranza ε ("epsilon") è possibile valutare la bontà di un'approssimazione tramite la seguente formula:

$$\left| \frac{f_n(x) - f_{n-1}(x)}{f_n(x)} \right| < \varepsilon,$$

dove $f_n(x)$ è la sommatoria descritta precedentemente.

Nel file `esercizio1.cc` scrivere la definizione della funzione `approssimazione_cos` in grado di calcolare e ritornare, usando la sommatoria sopra descritta, un'approssimazione della funzione coseno, calcolato in `x`. Tale funzione deve terminare quando l'approssimazione calcolata rientra in una data tolleranza `eps`, oppure dopo un certo numero `max_iter` di passi.

Si utilizzino le seguenti funzioni della libreria `math.h`:

```
double fabs (double x);
double pow (double x, double y);
```

che restituiscono rispettivamente il valore assoluto di un numero reale `x` e il valore di `x` elevato alla `y`, e la funzione `fattoriale` data.

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

1 **esercizio1.cc**

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

double fattoriale (double x);
double approssimazione_cos (double x, double eps, int max_iter);

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

  cout << "Inserisci il valore x, espresso in radianti, in cui vuoi calcolare cos(x): \n";
  cin >> x;
  cout << "Inserisci la tolleranza eps: \n";
  cin >> eps;
  cout << "Inserisci il numero massimo di passi da eseguire: \n";
  cin >> n;

  cout << "L'approssimazione di cos(x) e': "
       << approssimazione_cos (x, eps, n) << endl;

  return(0);
}

double fattoriale (double x)
{
  double valore=1.0, n;

  for (n=x; n>0; n--)
    {
      valore=valore*n;
    }

  return(valore);
}

double approssimazione_cos (double x, double eps, int n) {

  double app = 0, controllo;
  int i=0;

  do {
    double old;

    old = app;
    app = app + (pow ((double) -1,i) * pow (x,(2*i)) / fattoriale(2*i));
    controllo = fabs((app - old) / app);
    i++;

  } while ((controllo >= eps)&&(i<n));
```

```
  return (app);
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `ricerca_binaria` che esegue una ricerca binaria per verificare se un intero letto da tastiera è contenuto in un array di interi. Sono inoltre dati l'header file `database.h` ed il file binario `database.o` contenente la definizione delle strutture dati e gli header delle funzioni `init` (che inizializza ed ordina un array di interi) e `stampa` usate nel main.

NOTA 1: **La funzione `ricerca_binaria` 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`.

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

2 **esercizio2.cc**

```
using namespace std;
#include "database.h"

// Dichiarare qui sotto la funzione ricerca_binaria

int ricerca_binaria_2(int array[], int lb, int ub, int elem);
int ricerca_binaria(database d, int elem);

int main () {
   database d;
   int val;

   init(d,20);
   stampa(d);

   cout << "Cerca numbero: ";
   cin >> val;

   int i = ricerca_binaria(d,val);

   if (i >= 0) {
      cout << "Trovato: " << d.array[i] << endl;
   } else {
      cout << "NON Trovato!" << endl;
   }
   return 0;
}

// Definire qui sotto la funzione inserisci_elemento

int ricerca_binaria_2(int array[], int lb, int ub, int elem) {
   if (lb > ub) {
      return -1;
   }

   int mid = (lb+ub)/2;

   if (array[mid] == elem) {
      return mid;
   } else if (array[mid] > elem) {
      return ricerca_binaria_2(array, lb, mid-1, elem);
   } else {
      return ricerca_binaria_2(array, mid+1, ub, elem);
   }
}

int ricerca_binaria(database d, int elem) {
   return ricerca_binaria_2(d.array, 0, d.size, elem);
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `ricerca_binaria` che esegue una ricerca binaria per verificare se un numero float letto da tastiera è contenuto in un array di float. Sono inoltre dati l'header file `database.h` ed il file binario `database.o` contenente la definizione delle strutture dati e gli header delle funzioni `init` (che inizializza ed ordina un array di float) e `stampa` usate nel main.

NOTA 1: **La funzione `ricerca_binaria` 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`.

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

2 **esercizio2.cc**

```
using namespace std;
#include "database.h"

// Dichiarare qui sotto la funzione ricerca_binaria

int ricerca_binaria_2(float array[], int lb, int ub, float elem);
int ricerca_binaria(database d, float elem);

int main () {
   database d;
   float val;

   init(d,20);
   stampa(d);

   cout << "Cerca numbero: ";
   cin >> val;

   int i = ricerca_binaria(d,val);

   if (i >= 0) {
      cout << "Trovato: " << d.array[i] << endl;
   } else {
      cout << "NON Trovato!" << endl;
   }
   return 0;
}

// Definire qui sotto la funzione inserisci_elemento

int ricerca_binaria_2(float array[], int lb, int ub, float elem) {
   if (lb > ub) {
      return -1;
   }

   int mid = (lb+ub)/2;

   if (array[mid] == elem) {
      return mid;
   } else if (array[mid] > elem) {
      return ricerca_binaria_2(array, lb, mid-1, elem);
   } else {
      return ricerca_binaria_2(array, mid+1, ub, elem);
   }
}

int ricerca_binaria(database d, float elem) {
   return ricerca_binaria_2(d.array, 0, d.size, elem);
}
```

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;
- `deinit` liberi la memoria utilizzata dalla 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 delle operazioni `print` e `deinit`, 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 esercizio3.cc

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

#include "queue.h"


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

void deinit (queue &q)
{
    node *n = q.head;
    while (n != NULL)
    {
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
    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;
- `deinit` liberi la memoria utilizzata dalla 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 delle operazioni `print` e `deinit`, 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).

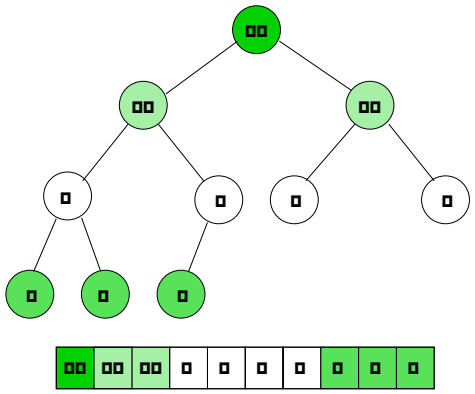
3 esercizio3.cc

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

#include "queue.h"


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

void deinit (queue &q)
{
    node *n = q.head;
    while (n != NULL)
    {
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
    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;
}
```

4 Un binary heap è una struttura dati che consiste in un albero binario quasi completo (i.e. riempito completamente su tutti i livelli tranne, eventualmente, l'ultimo che è riempito da sinistra a destra) dove il valore di un nodo figlio è minore o uguale a quello del nodo padre. Lo heap può essere efficentemente memorizzato con un array `A`, dove `A[0]` è il nodo radice e, se i è l'indice di un nodo, l'idice del padre `parent(i)`, del figlio sinistro `left(i)` e del figlio destro `right(i)` possono essere rispettivamente calcolati con $\lceil (i-1)/2 \rceil$, $2i+1$, $2i+2$.

La figura sottostante mostra un binary heap che contiene 10 elementi e l'array utilizzato per memorizzarlo.



Sono dati:

- il file `binaryheap.h`, contenente la definizione delle strutture dati e gli header delle funzioni
- il file binario `binaryheap.o`, contenente le funzioni `stampa_heap` e `rimuovi_elemento` usate nel main
- il file `esercizio4.cc`, contenente la funzione `main`.

Definire in `esercizio4.cc` la funzione `inserisci_elemento` che inserisce un elmento nello heap.

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

4 esercizio4.cc

```
using namespace std;
#include "binaryheap.h"

int main () {
   char res;
   binaryheap h;
   int n;

   build_heap(h,20);
   do {
      cout << "\nOperazioni possibili:\n"
           << "Inserisci (i): inserisci un elemento\n"
           << "Estrai (e): estrai l'elemento massimo\n"
           << "Stampa (s): stampa heap\n"
           << "Fine (f)\n";
      cout << "\nScegli operazione: ";
      cin >> res;
      switch (res) {
      case 'i':
         cin >> n;
         if (inserisci_elemento(h,n)==FAIL) {
            cout << "Heap pieno!\n";;
         }
         break;
      case 'e':
         if (estrai_massimo(h,n)==OK) {
            cout << n << endl;
         } else {
            cout << "Heap vuota\n";
         }
         break;
      case 's':
         stampa_heap(h);
         break;
      case 'f':
         break;
      default:
         cout << "Valore errato!\n";
      }
   } while (res != 'f');
}

// Definire qui sotto la funzione inserisci_elemento

static boolean pieno(binaryheap & h) {
   if (h.size == DIM) {
      return TRUE;
   }
   return FALSE;
}

retval inserisci_elemento(binaryheap & h, int elem) {
```

```
   if (pieno(h)==TRUE) {
      return FAIL;
   }
   h.size++;
int i = h.size;

while(i > 1 && h.array[i/2-1] < elem) {
    h.array[i-1] = h.array[i/2-1];
    i = i/2;
}
h.array[i-1] = elem;
   return OK;
}
```