Quinto Appello di Programmazione I

02 Settembre 2009
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 allochi dinamicamente una matrice di **interi** (le cui dimensioni ed elementi verranno acquisiti da tastiera), calcoli il **numero di elementi diversi da zero** contenuti nella matrice ed il **valore massimo**, liberando infine la memoria allocata.

In particolare, nel programma devono essere dichiarate e definite le seguenti tre subroutines:

(a) la funzione `alloca_matrice` che, presi come parametri il numero di righe e il numero di colonne della matrice, restituisca una matrice di interi allocando dinamicamente la memoria necessaria (NB: qui si effettui solo l'allocazione della matrice, senza acquisirne gli elementi);

(b) la procedura `calcola_nonzero_e_max` che presi come parametri la matrice e le sue dimensioni (righe e colonne) calcoli e restituisca, attraverso altri due parametri, rispettivamente il numero di elementi diversi da zero contenuti nella matrice ed il valore massimo trovato;

(c) la procedura `dealloca_matrice` che liberi correttamente tutta la memoria allocata precedentemente.

Il programma dovrà essere compilato utilizzando il file oggetto `inputmatrice.o` corrispondente al file `inputmatrice.h` incluso dal programma e nel quale è già definita la procedura `input_matrice` che si occupa di acquisire gli elementi della matrice.

Se ad esempio viene inserita la seguente matrice di 2 righe e 4 colonne

```
4 8 3 2
6 7 8 9
```

il programma dovrà stampare a video:

```
Numero di elementi diversi da zero: 8
Elemento massimo contenuto nella matrice: 9
```

e infine deallocherà la matrice. Se invece viene inserita la la seguente matrice di 3 righe e 3 colonne

```
 0 -2  0
-1  0 -1
 0 -4  0
```

il programma dovrà stampare a video:

```
Numero di elementi diversi da zero: 4
Elemento massimo contenuto nella matrice: 0
```

deallocando infine la memoria precedentemente allocata.

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

1 **esercizio1.cc**

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

#include "inputmatrice.h"

int **alloca_matrice (int righe, int colonne);
void calcola_nonzero_e_max (int **m, int righe, int colonne, int &nonzero, int &max_val);
void dealloca_matrice (int **m, int righe, int colonne);

int main ()
{
  int n_righe = 0, n_colonne =0;
  int **matrice = 0;

  // Acquisizione dimensioni della matrice
  cout << "Inserire il numero di righe della matrice da inserire: ";
  cin >> n_righe;
  cout << "Inserire il numero di colonne della matrice da inserire: ";
  cin >> n_colonne;

  // Allocazione matrice
  matrice = alloca_matrice(n_righe, n_colonne);

  // Acquisizione della matrice da tastiera
  cout << "Inserire i " << n_righe * n_colonne
       << " elementi interi della matrice: " << endl;
  input_matrice(matrice, n_righe, n_colonne);

  // Calcoli sulla matrice
  int nz, max;
  calcola_nonzero_e_max(matrice, n_righe, n_colonne, nz, max);

  cout << "Numero di elementi diversi da zero: " << nz << endl;
  cout << "Elemento massimo contenuto nella matrice: " << max << endl;

  // Deallocazione della matrice
  dealloca_matrice(matrice, n_righe, n_colonne);

  return 0;
}

int **alloca_matrice (int righe, int colonne)
{
  int **m = new int*[righe];
  for (int i = 0; i < righe; ++i)
    m[i] = new int[colonne];
  return m;
}

void calcola_nonzero_e_max (int **m, int righe, int colonne, int &nonzero, int &max_val)
{
```

```
  nonzero = 0;
  max_val = m[0][0];
  for (int i = 0; i < righe; ++i)
    for (int j = 0; j < colonne; ++j)
    {
      if (m[i][j]) ++nonzero;
      if (m[i][j] > max_val) max_val = m[i][j];
    }
}

void dealloca_matrice (int **m, int righe, int colonne)
{
  for (int i = 0; i < righe; ++i)
    delete [] m[i];
  delete [] m;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che allochi dinamicamente una matrice di **float** (le cui dimensioni ed elementi verranno acqusisti da tastiera), ne ricerchi l'**elemento minimo** e calcoli la **somma** di tutti gli elementi contenuti nella matrice, liberando infine la memoria allocata.

In particolare, nel programma devono essere dichiarate e definite le seguenti tre subroutines:

(a) la funzione `alloca_matrice` che, presi come parametri il numero di righe e il numero di colonne della matrice, restituisca una matrice di `float` allocando dinamicamente la memoria necessaria (NB: qui si effettui solo l'allocazione della matrice, senza acquisirne gli elementi);

(b) la procedura `calcola_min_e_somma` che presi come parametri la matrice e le sue dimensioni (righe e colonne) restituisca, attraverso altri due parametri, rispettivamente il valore minimo contenuto nella matrice e la somma di tutti gli elementi.

(c) la procedura `dealloca_matrice` che liberi correttamente tutta la memoria allocata precedentemente.

Il programma dovrà essere compilato utilizzando il file oggetto `inputmatrice.o` corrispondente al file `inputmatrice.h` incluso dal programma e nel quale è già definita la procedura `input_matrice` che si occupa di acquisire i valori reali della matrice.

Se ad esempio viene inserita la seguente matrice di `2` righe e `4` colonne

```
4.0 8.8 1.2 3.0
5.5 7.1 2.9 9.0
```

il programma dovrà stampare a video:

```
Elemento minimo contenuto nella matrice: 1.2
Somma di tutti gli elementi: 41.5
```

e infine deallocherà la matrice. Se invece viene inserita la la seguente matrice di `3` righe e `3` colonne

```
0 1 0
1 0 1
0 1 0
```

il programma dovrà stampare a video:

```
Elemento minimo contenuto nella matrice: 0
Somma di tutti gli elementi: 4
```

deallocando infine la memoria precedentemente allocata.

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

1 **esercizio1.cc**

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

#include "inputmatrice.h"

float **alloca_matrice (int righe, int colonne);
void calcola_min_e_somma (float **m, int righe, int colonne, float &min_val, float &somma);
void dealloca_matrice (float **m, int righe, int colonne);

int main ()
{
  int n_righe = 0, n_colonne =0;
  float **matrice = 0;

  // Acquisizione dimensioni della matrice
  cout << "Inserire il numero di righe della matrice da inserire: ";
  cin >> n_righe;
  cout << "Inserire il numero di colonne della matrice da inserire: ";
  cin >> n_colonne;

  // Allocazione matrice
  matrice = alloca_matrice(n_righe, n_colonne);

  // Acquisizione della matrice da tastiera
  cout << "Inserire i " << n_righe * n_colonne
       << " elementi reali della matrice: " << endl;
  input_matrice(matrice, n_righe, n_colonne);

  // Calcoli sulla matrice
  float min, sum;
  calcola_min_e_somma(matrice, n_righe, n_colonne, min, sum);

  cout << "Elemento minimo contenuto nella matrice: " << min << endl;
  cout << "Somma di tutti gli elementi: " << sum << endl;

  // Deallocazione della matrice
  dealloca_matrice(matrice, n_righe, n_colonne);

  return 0;
}

float **alloca_matrice (int righe, int colonne)
{
  float **m = new float*[righe];
  for (int i = 0; i < righe; ++i)
    m[i] = new float[colonne];
  return m;
}

void calcola_min_e_somma (float **m, int righe, int colonne, float &min_val, float &somma)
{
```

```
  min_val = m[0][0];
  somma = 0.0;
  for (int i = 0; i < righe; ++i)
    for (int j = 0; j < colonne; ++j)
    {
      somma += m[i][j];
      if (m[i][j] < min_val) min_val = m[i][j];
    }
}

void dealloca_matrice (float **m, int righe, int colonne)
{
  for (int i = 0; i < righe; ++i)
    delete [] m[i];
  delete [] m;
}
```

2 Nel file `esercizio2.cc` è contenuto un programa che, attraverso la funzione `search_it`, effettua in modo iterativo la ricerca binaria di un **numero** all'interno di un array ordinato di al più 100 **interi**.

Scrivere nel file `esercizio2.cc` la definizio dell'equivalente funzione `search_rec` che effettui la ricerca attraverso il medesimo procedimento binario, utilizzando però la **ricorsione**.

N.B. La funzione `search_rec` non può essere iterativa: al suo interno, non ci possono quindi essere cicli o chiamate a funzioni contenenti cicli.

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

2 **esercizio2.cc**

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

void leggi_interi (int [], int);

int search_it  (int [], int, int, int);
int search_rec (int [], int, int, int);

int main()
{
  int target, found_it, found_rec, dim;
  int v_interi [100];
  char risp;

  cout << "Inserisci il numero di interi da memorizzare: ";
  cin  >> dim;

  leggi_interi(v_interi, dim);

  do
  {
    cout << "Inserisci l'intero da cercare: ";
    cin  >> target;

    found_it  = search_it (v_interi, 0, dim-1, target);
    found_rec = search_rec(v_interi, 0, dim-1, target);

    cout << "ITERATIVA: ";
    if (found_it >= 0)
      cout << target << " si trova in posizione v_interi[" << found_it << "].\n";
    else
      cout << target << " non e' presente.\n";

    cout << "RICORSIVA: ";
    if (found_it >= 0)
      cout << target << " si trova in posizione v_interi[" << found_rec << "].\n";
    else
      cout << target << " non e' presente.\n";

    cout << "\nVuoi ripetere la ricerca? (s/n) ";
    cin  >> risp;
  }
  while (risp == 's');

  return 0;
}

void leggi_interi (int interi [100], int numero)
{
  for (int i = 0; i < numero; i++)
  {
    cout << "Inserire l'intero " << i+1 << " : ";
```

```
    cin  >> interi[i];
  }
}

int search_it (int a[], int left, int right, int target)
{
  while (left < right)
  {
    int middle = (left + right)/2;
    if (target > a[middle])
      left = middle + 1;
    else
      right = middle;
  }

  if (left == right)
  {
    if (a[left] == target)
      return left;
    else
      return -1;
  }

  // non puo' mai occorrere (left > right) salvo passaggio di parametri errato
  return -1;
}

int search_rec (int a[], int left, int right, int target)
{
  if (left == right)
  {
    return ((a[left] == target) ? left : -1);
  }
  else if (left < right)
  {
    int middle = (left + right)/2;
    if (target > a[middle])
      return search_rec (a, middle + 1, right, target);
    else
      return search_rec (a, left, middle, target);
  }
  // non puo' mai occorrere (left > right) salvo passaggio di parametri errato
  else
    return -1;
}
```

2 Nel file `esercizio2.cc` è contenuto un programa che, attraverso la funzione `search_it`, effettua in modo iterativo la ricerca binaria di un **carattere** all'interno di un array ordinato di al più 100 **caratteri**.

Scrivere nel file `esercizio2.cc` la definizio dell'equivalente funzione `search_rec` che effettui la ricerca attraverso il medesimo procedimento binario, utilizzando però la **ricorsione**.

N.B. La funzione `search_rec` non può essere iterativa: al suo interno, non ci possono quindi essere cicli o chiamate a funzioni contenenti cicli.

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

2 **esercizio2.cc**

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

void leggi_caratteri (char [], int);

int search_it  (char [], int, int, char);
int search_rec (char [], int, int, char);

int main()
{
  int found_it, found_rec, dim;
  char v_caratteri [100];
  char target;
  char risp;

  cout << "Inserisci il numero di caratteri da memorizzare: ";
  cin  >> dim;

  leggi_caratteri(v_caratteri, dim);

  do
  {
    cout << "Inserisci il carattere da cercare: ";
    cin  >> target;

    found_it  = search_it (v_caratteri, 0, dim-1, target);
    found_rec = search_rec(v_caratteri, 0, dim-1, target);

    cout << "ITERATIVA: ";
    if (found_it >= 0)
      cout << target << " si trova in posizione v_caratteri[" << found_it << "].\n";
    else
      cout << target << " non e' presente.\n";

    cout << "RICORSIVA: ";
    if (found_it >= 0)
      cout << target << " si trova in posizione v_caratteri[" << found_rec << "].\n";
    else
      cout << target << " non e' presente.\n";

    cout << "\nVuoi ripetere la ricerca? (s/n) ";
    cin  >> risp;
  }
  while (risp == 's');

  return 0;
}

void leggi_caratteri (char caratteri [100], int numero)
{
  for (int i = 0; i < numero; i++)
  {
```

```
    cout << "Inserire il carattere " << i+1 << " : ";
    cin  >> caratteri[i];
  }
}

int search_it (char a[], int left, int right, char target)
{
  while (left < right)
  {
    int middle = (left + right)/2;
    if (target > a[middle])
      left = middle + 1;
    else
      right = middle;
  }

  if (left == right)
  {
    if (a[left] == target)
      return left;
    else
      return -1;
  }

  // non puo' mai occorrere (left > right) salvo passaggio di parametri errato
  return -1;
}

int search_rec (char a[], int left, int right, char target)
{
  if (left == right)
  {
    return ((a[left] == target) ? left : -1);
  }
  else if (left < right)
  {
    int middle = (left + right)/2;
    if (target > a[middle])
      return search_rec (a, middle + 1, right, target);
    else
      return search_rec (a, left, middle, target);
  }
  // non puo' mai occorrere (left > right) salvo passaggio di parametri errato
  else
    return -1;
}
```

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.

N.B.: Ad eccezione delle operazioni `print` e `deinit`, si ricorda che ogni altra operazione sulla coda deve avere costo costante, indipendentemente dal numero di valori contenuti nella struttura dati.

VALUTAZIONE: questo esercizio vale 6 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');
```

```
    deinit(q);

    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   deinit  (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;
}

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 `stack_main.cc` è definita la funzione `main` che contiene un menu per gestire una pila di interi `long`. Scrivere, in un nuovo file `stack.cc`, le definizioni delle funzioni dichiarate nello header file `stack.h` in modo tale che:

- `init` inizializzi la pila;
- `deinit` liberi la memoria utilizzata dalla pila;
- `empty` restituisca `FALSE` se la pila contiene almeno un valore, e `TRUE` altrimenti;
- `push` inserisca il valore passato come parametro nella pila;
- `pop` elimini il valore in testa alla pila, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `top` legga il valore in testa alla pila 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 pila, nell'ordine in cui i valori contenuti ne verrebbero estratti.

La pila deve essere implementata internamente come una lista concatenata i cui elementi sono allocati dinamicamente.

N.B.: Ad eccezione delle operazioni `print` e `deinit`, si ricorda che ogni altra operazione sulla pila deve avere costo costante, indipendentemente dal numero di valori contenuti nella struttura dati.

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

3 stack_main.cc

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

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    long val;

    init(s);

    do
    {
        cout << "Operazioni possibili:\n"
             << "u : push\n"
             << "o : pop\n"
             << "t : top\n"
             << "p : print\n"
             << "e : esci\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'u':
            cout << "Valore da inserire? ";
            cin >> val;
            push(s, val);
            break;
          case 'o':
            if (! pop(s))
                cout << "Stack vuoto!\n";
            else
                cout << "Pop ok!\n";
            break;
          case 't':
            if (! top(s, val))
                cout << "Stack vuoto!\n";
            else
                cout << "Top: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto dello stack: ";
            print(s);
            break;
          default:
            if (scelta != 'e') {
                cout << "Operazione non valida: " << scelta << endl;
            }
        }
```

```
    } while (scelta != 'e');

    deinit(s);

    return 0;
}
```

3 stack.h

```
#ifndef STACK_H
 #define STACK_H


struct node {
  long val;
  node *next;
};

typedef node *stack;


enum retval {FALSE = 0, TRUE = 1};


void   init   (stack &s);
void   deinit (stack &s);
void   push   (stack &s, long val);
retval pop    (stack &s);
retval empty  (const stack &s);
retval top    (const stack &s, long &result);
void   print  (const stack &s);


#endif // STACK_H
```

3 soluzione_A32.cc

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

#include "stack.h"


void init (stack &s)
{
    s = NULL;
}

void deinit (stack &s)
{
    node *n = s;
    while (n != NULL)
    {
```

```
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
    s = NULL;
}

retval empty (const stack &s)
{
    return (s == NULL ? TRUE : FALSE);
}

void push (stack &s, long val)
{
    node *n = new node;

    n->val = val;
    n->next = s;
    s = n;
}

retval pop (stack &s)
{
    if (empty(s))
      return FALSE;

    node *first = s;
    s = s->next;
    delete first;

    return TRUE;
}

retval top (const stack &s, long &result)
{
    if (empty(s))
      return FALSE;

    result = s->val;
    return TRUE;
}

void print (const stack &s)
{
    node *n = s;
    while (n != NULL)
    {
        cout << n->val << " ";
        n = n->next;
    }
    cout << endl;
}
```