Quarto Appello di Programmazione I

09 Luglio 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` la dichiarazione e la definzione della funzione `ricerca` che effettua una ricerca su una matrice di interi. Tale funzione dovrà restituire il numero di volte che un dato valore intero passato come parametro risulta presente nella matrice inserita (quindi `0` nel caso in cui tale valore non sia presente). Inoltre, la funzione `ricerca` dovrà restituire le coordinate indicanti la posizione della prima istanza dell'intero trovata nella matrice (procedendo prima per righe e poi per colonne).

Se ad esempio viene inserita la seguente matrice:

```
2 5 0 1 2 3
4 5 6 7 8 9
```

e viene richiesto di ricercare l'intero `5`, il programma fornirà a video il seguente risultato:

```
L'intero 5 e' presente 2 volte nella matrice inserita.
La prima istanza dell'intero 5 e' presente alle coordinate 0,1.
```

Se invece viene inserita la matrice:

```
0 1 2
3 4 5
6 7 8
```

e viene richiesto di ricercare il valore `9`, il programma stamperà a video:

```
L'intero 9 non esiste nella matrice inserita.
```

NOTA: Il programma include il file `inputmatrice.h` (nel quale e' definita la procedura `input_matrice`), e dovrà quindi essere compilato utilizzando anche il corrispondente file oggetto `inputmatrice.o`.

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

1 **esercizio1.cc**

```
#include <iostream>
#include "inputmatrice.h"

using namespace std;


int ricerca (int matrice[][MaxDim], int dim_righe, int dim_colonne, int target, int & riga_trov


int main ()
{
  int matrice [MaxDim][MaxDim];
  int dim_righe, dim_colonne;
  int target, trovato, riga_trovato, colonna_trovato;

  cout << "Inserire il numero di righe della matrice da inserire: ";
  cin >> dim_righe;

  cout << "Inserire il numero di colonne della matrice da inserire: ";
  cin >> dim_colonne;

  input_matrice(matrice, dim_righe, dim_colonne);

  cout << "Inserire l'intero da trovare nella matrice inserita: ";
  cin >> target;

  trovato = ricerca (matrice, dim_righe, dim_colonne, target, riga_trovato, colonna_trovato);

  if (trovato == 0)
    cout << "L'intero " << target << " non esiste nell matrice inserita." << endl;
  else {
    cout << "L'intero " << target << " e' presente " << trovato << " volte nella matrice inseri
    cout << "La prima istanza dell'intero " << target << " e' presente alle coordinate " << rig
  }
  return 0;
}

int ricerca (int matrice[][MaxDim], int dim_righe, int dim_colonne, int target, int & riga_trov
{
  int trovato = 0;

  for (int i = 0; i < dim_righe; i++) {
    for (int j = 0; j < dim_colonne; j++) {
     if (matrice[i][j] == target) {
        trovato++;
        if (trovato == 1) {
          riga_trovato = i;
          colonna_trovato = j;
        }
      }
    }
  }
```

```
  return (trovato);
}
```

1 Scrivere nel file `esercizio1.cc` la dichiarazione e la definzione della funzione `search` che effettua una ricerca su una matrice di numeri reali. Tale funzione dovrà restituire il numero di volte che un dato valore reale passato come parametro risulta presente nella matrice inserita (quindi `0` nel caso in cui tale valore non sia presente). Inoltre, la funzione `search` dovrà restituire le coordinate indicanti la posizione della prima istanza del numero reale trovata nella matrice (procedendo prima per righe e poi per colonne).

Se ad esempio viene inserita la seguente matrice:

```
2.2 5.0 0.0 1.7
4.3 5.0 6.0 5.5
```

e viene richiesto di ricercare il numero reale `5.0`, il programma fornirà a video il seguente risultato:

```
Il numero 5 e' presente 2 volte nella matrice inserita.
La prima istanza del numero 5 e' presente alle coordinate 0,1.
```

Se invece viene inserita la matrice:

```
0.0 1.1 2.2
3.3 4.4 5.5
6.6 7.7 8.8
```

e viene richiesto di ricercare il valore `9.0`, il programma stamperà a video:

```
Il numero 9 non esiste nella matrice inserita.
```

NOTA: Il programma include il file `inputmatrice.h` (nel quale e' definita la procedura `input_matrice`), e dovrà quindi essere compilato utilizzando anche il corrispondente file oggetto `inputmatrice.o`.

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

1 **esercizio1.cc**

```
#include <iostream>
#include "inputmatrice.h"

using namespace std;


int search (float matrice[][MaxDim], int dim_righe, int dim_colonne, float target, int & riga_t


int main ()
{
  float matrice [MaxDim][MaxDim];
  int dim_righe, dim_colonne;
  int trovato, riga_trovato, colonna_trovato;
  float target;

  cout << "Inserire il numero di righe della matrice da inserire: ";
  cin >> dim_righe;

  cout << "Inserire il numero di colonne della matrice da inserire: ";
  cin >> dim_colonne;

  input_matrice(matrice, dim_righe, dim_colonne);

  cout << "Inserire il numero da trovare nella matrice inserita: ";
  cin >> target;

  trovato = search (matrice, dim_righe, dim_colonne, target, riga_trovato, colonna_trovato);

  if (trovato == 0)
    cout << "Il numero " << target << " non esiste nell matrice inserita." << endl;
  else {
    cout << "Il numero " << target << " e' presente " << trovato << " volte nella matrice inser
    cout << "La prima istanza del numero " << target << " e' presente alle coordinate " << riga
  }
  return 0;
}


int search (float matrice[][MaxDim], int dim_righe, int dim_colonne, float target, int & riga_t
{
  int trovato = 0;

  for (int i = 0; i < dim_righe; i++) {
    for (int j = 0; j < dim_colonne; j++) {
     if (matrice[i][j] == target) {
        trovato++;
        if (trovato == 1) {
          riga_trovato = i;
          colonna_trovato = j;
        }
      }
```

```
    }
  }

  return (trovato);
}
```

2 Nel file `esercizio2.cc` è definita la funzione **ricorsiva**

```
int f_rec(char *s, char c1, char c2);
```

Scrivere, nello stesso file, la definizione della funzione **iterativa**

```
int f_iter(char *s, char c1, char c2);
```

il cui comportamento deve essere identico a quello della corrispondente versione ricorsiva `f_rec`.

NOTA: 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**

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


int f_rec(char *s, char c1, char c2);
int f_iter(char *s, char c1, char c2);

int main()
{
    const int dim = 100;
    char parola1[dim];
    char parola2[dim];
    char c1, c2;

    cout << "Inserisci la parola: ";
    cin >> parola1;
    cout << "Inserisci carattere 1: ";
    cin >> c1;
    cout << "Inserisci carattere 2: ";
    cin >> c2;

    strncpy(parola2, parola1, dim-1);
    parola2[dim-1] = '\0';

    int r1 = f_rec(parola1, c1, c2);
    int r2 = f_iter(parola2, c1, c2);

    cout << "Risultato f_rec:  " << parola1 << ", " << r1 << endl;
    cout << "Risultato f_iter: " << parola2 << ", " << r2 << endl;

    return(0);
}


int f_rec(char *s, char c1, char c2)
{
    int res = -1;
    if (*s != '\0') {
        if (*s == c1) {
            *s = c2;
            res = 0;
        } else {
            int r = f_rec(s+1, c1, c2);
            if (r >= 0) {
                res = r+1;
            }
        }
    }
    return res;
}
```

```
// scrivere qui sotto la definizione della funzione f_iter

int f_iter(char *s, char c1, char c2)
{
    int res = -1;
    for (int i = 0; s[i] != '\0' && res == -1; ++i) {
        if (s[i] == c1) {
            s[i] = c2;
            res = i;
        }
    }
    return res;
}
```

2 Nel file `esercizio2.cc` è definita la funzione **ricorsiva**

```
int fun_ricorsiva(char *s, char c1, char c2);
```

Scrivere, nello stesso file, la definizione della funzione **iterativa**

```
int fun_iterativa(char *s, char c1, char c2);
```

il cui comportamento deve essere identico a quello della corrispondente versione ricorsiva `fun_ricorsiva`.

NOTA: 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**

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


int fun_ricorsiva(char *s, char c1, char c2);
int fun_iterativa(char *s, char c1, char c2);

int main()
{
    const int dim = 100;
    char parola1[dim];
    char parola2[dim];
    char c1, c2;

    cout << "Inserisci la parola: ";
    cin >> parola1;
    cout << "Inserisci carattere 1: ";
    cin >> c1;
    cout << "Inserisci carattere 2: ";
    cin >> c2;

    strncpy(parola2, parola1, dim-1);
    parola2[dim-1] = '\0';

    int r1 = fun_ricorsiva(parola1, c1, c2);
    int r2 = fun_iterativa(parola2, c1, c2);

    cout << "Risultato fun_ricorsiva: " << parola1 << ", " << r1 << endl;
    cout << "Risultato fun_iterativa: " << parola2 << ", " << r2 << endl;

    return(0);
}


int fun_ricorsiva(char *s, char c1, char c2)
{
    int res = 1;
    if (*s != '\0') {
        if (*s == c2) {
            *s = c1;
            res = 0;
        } else {
            int r = fun_ricorsiva(s+1, c1, c2);
            if (r <= 0) {
                res = r-1;
            }
        }
    }
    return res;
}
```

```
// scrivere qui sotto la definizione della funzione f_iter

int fun_iterativa(char *s, char c1, char c2)
{
    int res = 1;
    for (int i = 0; s[i] != '\0' && res == 1; ++i) {
        if (s[i] == c2) {
            s[i] = c1;
            res = -i;
        }
    }
    return res;
}
```

3 Nel file `queue_main.cc` è definita la funzione `main` che contiene un menu per gestire una **coda di interi**. 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 nella coda l'elemento passato come parametro, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `dequeue` rimuova l'elemento 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 (senza modificarla) 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, nell'ordine in cui i valori contenuti ne verebbero estratti.

La coda deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata attraverso il parametro `dim` passato alla funzione `init`.

Nota: Si ricorda che, fatta eccezione per l'operazione `print`, ogni altra operazione sulla coda deve avere costo costante, indipendentemente dal numero di valori interi conenuti 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 res;
    int num;
    queue q;
    const int maxdim = 100;

    init(q, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << "Enqueue (e)\n"
             << "Dequeue (d)\n"
             << "First   (f)\n"
             << "Print   (p)\n"
             << "Quit    (q)\n";
        cin >> res;
        switch (res) {
        case 'e':
            cout << "Valore: ";
            cin >> num;
            if (enqueue(num, q) == FALSE)
              cout << "Coda piena\n";
            else
              cout << "Valore inserito\n";
            break;
        case 'd':
            if (dequeue(q) == FALSE)
              cout << "Coda vuota\n";
            else
              cout << "Valore rimosso\n";
            break;
        case 'f':
            if (first(num, q) == FALSE)
              cout << "Coda vuota\n";
            else
              cout << "First = " << num << endl;
            break;
        case 'p':
            print(q);
            break;
        case 'q':
            break;
        default:
            cout << "Valore errato!\n";
        }
    } while (res != 'q');
```

```
    deinit(q);

    return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_QUEUE_H_INCLUDED
 #define STRUCT_QUEUE_H_INCLUDED

enum retval {FALSE, TRUE};

struct queue {
  int testa, coda;
  int dimensione;
  int *elementi;
};

void   init    (queue &q, int dim);
void   deinit  (queue &q);
retval enqueue (int val, queue &q);
retval dequeue (queue &q);
retval first   (int &val, const queue &q);
void   print   (const queue &q);

#endif  // STRUCT_QUEUE_H_INCLUDED
```

3 **queue.cc**

```
#include "queue.h"
#include <iostream>

using namespace std;


static bool is_empty (const queue &q)
{
  return q.coda == q.testa;
}

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

static bool is_full (const queue &q)
{
  return next(q.coda, q.dimensione) == q.testa;
}

void init (queue &q, int dim)
{
```

```
  q.coda = q.testa = 0;
  q.dimensione = dim;
  q.elementi = new int[dim];
}

void deinit (queue &q)
{
  delete[] q.elementi;
}

retval enqueue (int val, queue &q)
{
  if (is_full(q))
    return FALSE;

  q.elementi[q.coda] = val;
  q.coda = next(q.coda, q.dimensione);
  return TRUE;
}

retval dequeue (queue &q)
{
  if (is_empty(q))
    return FALSE;

  q.testa = next(q.testa, q.dimensione);
  return TRUE;
}

retval first (int &val, const queue &q)
{
  if (is_empty(q))
    return FALSE;

  val = q.elementi[q.testa];
  return TRUE;
}

void print (const queue &q)
{
  int i;
  for (i = q.testa; i < q.coda; i = next(i, q.dimensione))
  {
    cout << q.elementi[i] << " ";
  }
  cout << endl;
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menu per gestire una **pila di interi**. 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 per contenere al più `dim-1` elementi;
- `deinit` liberi la memoria utilizzata dalla pila;
- `push` inserisca nella pila l'elemento passato come parametro, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `pop` rimuova l'elemento in cima alla pila, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `top` legga il valore in cima alla pila (senza modificarla) 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 (senza modificarla), nell'ordine in cui i valori contenuti ne verebbero estratti.

La pila deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata attraverso il parametro `dim` passato alla funzione `init`.

Nota: Si ricorda che, fatta eccezione per l'operazione `print`, ogni altra operazione sulla pila deve avere costo costante, indipendentemente dal numero di valori interi conenuti nella struttura dati.

VALUTAZIONE: questo esercizio vale 7 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 res;
    int val;
    stack s;
    const int maxdim = 100;

    init(s, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << " Push (u)\n"
             << " Pop (o)\n"
             << " Top (t)\n"
             << " Print (p)\n"
             << " Fine (f)\n";
        cin >> res;
        switch (res) {
        case 'u':
            cout << "Val? : ";
            cin >> val;
            if (push(val, s) == FALSE)
              cout << "Stack pieno\n";
            else
              cout << "Valore inserito\n";
            break;
        case 'o':
            if (pop(s) == FALSE)
              cout << "Stack vuoto\n";
            else
              cout << "Valore rimosso\n";
            break;
        case 't':
            if (top(val, s) == FALSE)
              cout << "Stack vuoto!\n";
            else
              cout << "Top = " << val << endl;
            break;
        case 'p':
            print(s);
            break;
        case 'f':
            break;
        default:
            cout << "Opzione errata\n";
        }
    } while (res != 'f');

    deinit(s);
```

```
    return 0;
}
```

3 **stack.h**

```
#ifndef STRUCT_STACK_H_INCLUDED
 #define STRUCT_STACK_H_INCLUDED

enum retval {FALSE, TRUE};

struct stack {
  int indice;
  int dimensione;
  int *elementi;
};

void   init   (stack &s, int dim);
void   deinit (stack &s);
retval push   (int val, stack &s);
retval pop    (stack &s);
retval top    (int &val, const stack &s);
void   print  (const stack &s);

#endif  // STRUCT_STACK_H_INCLUDED
```

3 **stack.cc**

```
#include "stack.h"
#include <iostream>

using namespace std;


static bool is_empty (const stack &s)
{
  return s.indice == 0;
}

static bool is_full (const stack &s)
{
  return s.indice == s.dimensione;
}

void init (stack &s, int dim)
{
  s.indice = 0;
  s.dimensione = dim;
  s.elementi = new int[dim];
}

void deinit (stack &s)
{
  delete[] s.elementi;
```

```
}

retval push (int val, stack &s)
{
  retval res;
  if (is_full(s))
    res = FALSE;
  else
  {
    s.elementi[s.indice++] = val;
    res = TRUE;
  }
  return res;
}

retval pop (stack &s)
{
  retval res;
  if (is_empty(s))
    res = FALSE;
  else
  {
    s.indice--;
    res = TRUE;
  }
  return res;
}

retval top (int &val, const stack &s)
{
  retval res;
  if (is_empty(s))
    res = FALSE;
  else
  {
    val = s.elementi[s.indice-1];
    res = TRUE;
  }
  return res;
}

void print (const stack &s)
{
  int i;
  for (i = 0; i < s.indice; i++)
  {
    cout << s.elementi[i] << " ";
  }
  cout << endl;
}
```