Primo Appello di Programmazione I

09 Giugno 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 Nel file `esercizio1.cc`‘e contenuto un programma che, preso come argomento del `main` il nome di un file, legge da tale file gli elementi di una matrice di **interi** allocandola dinamicamente e ne stampi a video il contenuto, liberando infine la memoria allocata.

In particolare, il file indicato come argomento conterrà una sequenza di valori **interi** (separati tra loro da caratteri di spaziatura) di cui, nell'ordine, il primo valore rappresenta il **numero delle RIGHE** della matrice, il secondo il **numero delle COLONNE** ed i restanti valori rappresentano gli **elementi della matrice**.

Nel programma devono essere essere definite le seguenti funzioni:

(a) la funzione `input_matrice` che preso come parametro il nome del file (una stringa) ritorni la matrice di numeri **interi** allocata dinamicamente, riempendola con i valori letti da file secondo la struttura sopra descritta;

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

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

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

il comando: `./a.out esempio_cifre`
allocherà una matrice di **interi** costituita da 2 righe, ognuna di 5 colonne, in cui caricherà i valori da 0 a 9 contenuti nel file, stampando a video:

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

e infine deallocherà la matrice. Dato invece il file `esempio_lost` contenente:

```
3 2 4 8 15 16 23 42
```

il comando: `./a.out esempio_lost`
allocherà una matrice di **interi** costituita da 3 righe, ognuna di 2 colonne, in cui caricherà i valori da 4 a 42 contenuti nel file, stampando a video:

```
4 8
15 16
23 42
```

e infine deallocherà la matrice precedentemente allocata.

NOTA: Per semplicità si assuma che vengano sempre indicati al programma file corretti, contenenti numeri di righe e colonne della matrice sempre maggiori di zero e contenenti sempre in numero esatto tutti gli elementi della matrice. Si ricorda che i file stream utilizzati all'interno del codice devono essere correttamente chiusi prima della fine del programma.

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 "outmatrice.h"

int **input_matrice (char *nome_file, int &righe, int &colonne);
void dealloca_matrice (int **matrice, int num_r, int num_c);

int main (int argc, char* argv[])
{
  if (argc != 2)
  {
    cout << "Eseguire: " << argv[0] << " <nome-file>" << endl;
    return 1;
  }

  int n_righe = 0, n_colonne =0;
  int **matrice = 0;

  // Acquisizione matrice da file
  matrice = input_matrice(argv[1], n_righe, n_colonne);
  // Stampa a video della matrice
  output_matrice(matrice, n_righe, n_colonne);
  // Deallocazione della matrice
  dealloca_matrice(matrice, n_righe, n_colonne);

  return 0;
}

int **input_matrice (char *nome_file, int &righe, int &colonne)
{
  fstream f_in;
  f_in.open(nome_file, ios::in);

  f_in >> righe;
  f_in >> colonne;
  int **m = new int*[righe];

  for (int i = 0; i < righe; ++i)
  {
    m[i] = new int[colonne];
    for (int j = 0; j < colonne; ++j)
      f_in >> m[i][j];
  }
  f_in.close();

  return m;
}

void dealloca_matrice (int **m, int righe, int colonne)
{
```

```
  for (int i = 0; i < righe; ++i)
    delete m[i];
  delete m;
}
```

1 Nel file `esercizio1.cc`‘e contenuto un programma che, preso come argomento del `main` il nome di un file, legge da tale file gli elementi di una matrice di **float** allocandola dinamicamente e ne stampa a video il contenuto, liberando infine la memoria allocata.

In particolare, il file indicato come argomento conterrà una sequenza di valori **reali** (separati tra loro da caratteri di spaziatura) di cui, nell'ordine, il primo valore rappresenta il **numero delle COLONNE** della matrice, il secondo il **numero delle RIGHE** ed i restanti valori rappresentano gli **elementi della matrice**.

Nel programma devono essere essere definite le seguenti funzioni:

(a) la funzione `input_matrice` che preso come parametro il nome del file (una stringa) ritorni la matrice di **float** allocata dinamicamente, riempendola con i valori letti da file secondo la struttura sopra descritta;

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

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

```
3 2 0.4 8.0 1.5 1.6 2.3 42.0
```

il comando: `./a.out esempio_lost`
allocherà una matrice di **float** costituita da 2 righe, ognuna di 3 colonne, in cui caricherà i valori contenuti nel file, stampando a video:

```
0.4 8 1.5
1.6 2.3 42
```

e infine deallocherà la matrice. Dato invece il file `esempio_cifre` contenente:

```
2 4 1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8
```

il comando: `./a.out esempio_cifre`
allocherà una matrice di **float** costituita da 4 righe, ognuna contenente 2 elementi, in cui caricherà i valori contenuti nel file, stampando a video:

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

e infine deallocherà la matrice precedentemente allocata.

NOTA: Per semplicità si assuma che vengano sempre indicati al programma file corretti, contenenti numeri di righe e colonne della matrice sempre maggiori di zero e contenenti sempre in numero esatto tutti gli elementi della matrice. Si ricorda che i file stream utilizzati all'interno del codice devono essere correttamente chiusi prima della fine del programma.

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 "outmatrice.h"

float **input_matrice (char *nome_file, int &righe, int &colonne);
void dealloca_matrice (float **matrice, int num_r, int num_c);

int main (int argc, char* argv[])
{
  if (argc != 2)
  {
    cout << "Eseguire: " << argv[0] << " <nome-file>" << endl;
    return 1;
  }

  int n_righe = 0, n_colonne =0;
  float **matrice = 0;

  // Acquisizione matrice da file
  matrice = input_matrice(argv[1], n_righe, n_colonne);
  // Stampa a video della matrice
  output_matrice(matrice, n_righe, n_colonne);
  // Deallocazione della matrice
  dealloca_matrice(matrice, n_righe, n_colonne);

  return 0;
}

float **input_matrice (char *nome_file, int &righe, int &colonne)
{
  fstream f_in;
  f_in.open(nome_file, ios::in);

  f_in >> colonne;
  f_in >> righe;
  float **m = new float*[righe];

  for (int i = 0; i < righe; ++i)
  {
    m[i] = new float[colonne];
    for (int j = 0; j < colonne; ++j)
      f_in >> m[i][j];
  }
  f_in.close();

  return m;
}

void dealloca_matrice (float **m, int righe, int colonne)
{
```

```
  for (int i = 0; i < righe; ++i)
    delete m[i];
  delete m;
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione `search` che effettua una ricerca **ricorsiva** su un array di al più 100 interi. Tale funzione dovrà trovare il primo multiplo intero di un numero passato come parametro all'interno di un array passato anch'esso come parametro e dovrà ritornare la posizione in cui si trova questo multiplo, -1 altrimenti.

NOTA: **La funzione deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

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

2 **esercizio2.cc**

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

void memorizza_array (int [], int);

//Inserire qui la dichiarazione della funzione search

int search(int [], int, int, int);

int main()
{
  int target, found, loc, dim;
  int array [100];
  char risp;

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

  memorizza_array (array, dim);

  do
    {
      cout << "Inserisci l'intero di cui cercare il primo multiplo: ";
      cin >> target;

      found = search (array, target, dim, 0);

      if (found!=-1) {
cout <<"Il primo multiplo di " << target << " trovato e' "<< array[found] << endl;
cout <<"e si trova nella posizione [" << found << "].\n";
      }
      else cout <<"Non e' stato trovato nessun multiplo di " << target << "\n";

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

  return 0;
}


void memorizza_array (int array [100], int dim)
{
  for (int i=0; i<dim; i++)
    {
      cout << "Inserire intero " << i+1 <<" : ";
      cin >> array [i];
    }
}
```

```
//Inserire qui la definizione della funzione search

int search (int array[], int target, int dim, int location)
{
   if (location >= dim)
     return -1;
   else
   {
     if (array[location] % target == 0)
       return location;
     else
       return search (array, target, dim, location+1);
   }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione `ricerca` che effettua una ricerca **ricorsiva** su un array di al più 100 interi. Tale funzione dovrà trovare il primo multiplo intero di un numero passato come parametro all'interno di un array passato anch'esso come parametro e dovrà ritornare la posizione in cui si trova questo multiplo, -1 altrimenti.

NOTA: **La funzione deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

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

2 **esercizio2.cc**

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

void memorizza_array (int [], int);

//Inserire qui la dichiarazione della funzione ricerca

int ricerca(int [], int, int, int);

int main()
{
  int obiettivo, trovato, loc, dim;
  int array [100];
  char risp;

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

  memorizza_array (array, dim);

  do
    {
      cout << "Inserisci l'intero di cui cercare il primo multiplo: ";
      cin >> obiettivo;

      trovato = ricerca (array, obiettivo, dim, 0);

      if (trovato!=-1) {
cout <<"Il primo multiplo di " << obiettivo << " trovato e' "<< array[trovato] << endl;
cout <<"e si trova nella posizione [" << trovato << "].\n";
      }
      else cout <<"Non e' stato trovato nessun multiplo di " << obiettivo << "\n";

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

  return 0;
}


void memorizza_array (int array [100], int dim)
{
  for (int i=0; i<dim; i++)
    {
      cout << "Inserire intero " << i+1 <<" : ";
      cin >> array [i];
    }
}
```

```
//Inserire qui la definizione della funzione ricerca

int ricerca (int array[], int obiettivo, int dim, int location)
{
   if (location >= dim)
     return -1;
   else
   {
     if (array[location] % obiettivo == 0)
       return location;
     else
       return ricerca (array, obiettivo, dim, location+1);
   }
}
```

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

- `init` inizializzi l'albero;
- `empty` controlli se l'albero è vuoto, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `insert` inserisca l'elemento passato come parametro nell'albero, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti (cioè se l'elemento è già presente). L'albero deve essere ordinato in maniera **crescente**. Esempio: l'inserimento dei seguenti valori:

  ```
  e  a  u
  ```

  deve produrre il seguente albero:

  ```
        e
     a     u
  ```

- `search` cerchi nell'albero l'elemento passato in input, resituendo `TRUE` se l'elemento è presente, e `FALSE` altrimenti;
- `print` stampi a video il contenuto dell'albero, in ordine **crescente**. Esempio: l'albero qui sopra deve essere stampato come:

  ```
  a  e  u
  ```

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

3 albero_main.cc

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

int main()
{
    char res;
    Tree tree;
    char val;
    init(tree);
    do {
        cout << "\nOperazioni possibili:\n"
             << " Inserimento (i)\n"
             << " Ricerca (r)\n"
             << " Stampa ordinata (s)\n"
             << " Fine (f)\n";
        cout << "Operazione selezionata: ";
        cin >> res;
        switch (res) {
        case 'i':
            cout << "Valore : ";
            cin >> val;
            if (insert(tree, val) == FALSE) {
                cout << "Valore gia` presente!" << endl;
            }
            break;
        case 'r':
            cout << "Valore: ";
            cin >> val;
            if (search(tree, val) == TRUE) {
                cout << "Valore presente: " << val << endl;
            } else {
                cout << "Valore non presente" << endl;
            }
            break;
        case 's':
            if (empty(tree) == TRUE) {
                cout << "Albero vuoto!" << endl;
            } else {
                print(tree);
            }
            break;
        case 'f':
            break;
        default:
            cout << "Opzione errata\n";
        }
    } while (res != 'f');

    return 0;
}
```

3 **albero.h**

```
// -*- C++ -*-
#ifndef ALBERO_H
#define ALBERO_H

struct Node {
    char val;
    Node *left;
    Node *right;
};

typedef Node * Tree;

enum boolean { FALSE, TRUE };

void init(Tree &t);
boolean empty(const Tree &t);
boolean insert(Tree &t, char val);
boolean search(const Tree &t, char val);
void print(const Tree &t);

#endif
```

3 **soluzione_A31.cc**

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

void init(Tree &t)
{
    t = NULL;
}


boolean empty(const Tree &t)
{
    return (t == NULL) ? TRUE : FALSE;
}


boolean insert(Tree &t, char val)
{
    // caso base
    if (empty(t) == TRUE) {
        t = new Node;
        t->val = val;
        t->left = t->right = NULL;
        return TRUE;
    }
    // caso ricorsivo. Controllo se scendere a sinistra o a destra
    if (val < t->val) {
        // scendo a sinistra
```

```
        return insert(t->left, val);
    } else if (val > t->val) {
        // scendo a destra
        return insert(t->right, val);
    } else {
        // elemento gia' presente, restituisco false
        return FALSE;
    }
}



boolean search(const Tree &t, char val)
{
    if (empty(t) == TRUE) {
        return FALSE;
    } else if (val == t->val) {
        return TRUE;
    } else if (val < t->val) {
        // scendo a sinistra
        return search(t->left, val);
    } else {
        // scendo a destra
        return search(t->right, val);
    }
}


void print(const Tree &t)
{
    if (empty(t) == FALSE) {
        // prima stampo gli elementi minori di t->val (cioe' quelli a sx)
        print(t->left);
        // poi stampo t->val
        cout << t-> val << ' ';
        // poi stampo gli elementi maggiori (cioe' quelli a dx)
        print(t->right);
    }
}
```

4 Si definisca una funzione **ricorsiva** "`prodotto`" che prenda in ingresso due numeri interi positivi n e m e restituisca un valore intero che rappresenti il prodotto $n \cdot m$, 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 `succpred.h`:

```
#ifdef  PREDSUCC_H
#define 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);
#endif
```

e il corrispondente file `succpred.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 prodotto

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 << " = " << prodotto(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.

4 **esercizio4.cc**

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

// introdurre qui la definizione della funzione prodotto
int somma (int n, int m) {
  int res;
  if (m==0)
    res = n;
  else
    res = succ(somma(n,pred(m)));
// Soluzione alternativa:
//  res = somma(succ(n),pred(m));
}

int prodotto (int n, int m) {
  int res;
  if (m==0)
    res = 0;
  else
    res = somma(n,prodotto(n,pred(m)));
  return res;
}

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 << " = " << prodotto(n,m) << endl;
  }
  while (n >= 0 && m >= 0);
}
```