Primo Appello di Programmazione I

13 Gennaio 2010
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` sono presenti le funzioni `stampa_array` e `main` definite come segue:

- la funzione `stampa_array` prende come parametro un array di interi e ne stampa a video il contenuto;
- la funzione `main` prende da standard input una parola e la memorizza in un array di caratteri chiamato `parola`, invoca poi la funzione `calcola_frequenza` ed infine chiama la funzione `stampa_array`.

Per semplicità supponiamo che la parola data in input sia composta solamente da caratteri minuscoli.

Realizzare la funzione `calcola_frequenza` che, preso in input un array di caratteri `parola` contenente una parola scritta in caratteri minuscoli, restituisca un array di interi contenente la frequenza delle lettere dell'alfabeto presenti nella parola data. In particolare ogni cella dell'array di interi deve corrispondere ad una lettera minuscola dell'alfabeto e deve contere il numero di lettere di questo tipo presenti in `parola`: ad esempio il primo elemento dell'array corrisponderà alla lettera 'a' dell'alfabeto e conterrà il numero di 'a' contenute nella parola data. L'array di interi deve quindi avere 26 caselle.

Se ad esempio la parola memorizzata in `parola` è *zibibbo* la funzione `calcola_frequenza` deve restituire il seguente array:

0	3	0	0	0	0	0	0	2	0	0	0	0	0	1	0	...	0	1

che corrisponde a dire: "Nella parola data ci sono: 0 lettere 'a', 3 lettere 'b', 0 lettere 'c', 0 lettere 'd', ..., 2 lettere 'i', ... ed 1 lettera 'z'".

NOTA: all'interno di questo programma non è ammesso l'utilizzo di variabili globali o di tipo `static`, ad eccezione di quelle già presenti.

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

1 **esercizio1.cc**

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

const int dim = 100;

void stampa_array (int []);
int* calcola_frequenza (char []);

int main(){

  char parola [dim];
  int * frequenza;

  cout << "Inserisci la parola (lettere minuscole): ";
  cin >> parola;

  frequenza = calcola_frequenza(parola);

  cout << "Nella parola data ci sono:" << endl;
  stampa_array(frequenza);

  return(0);
}

void stampa_array (int vett[])
{
  for(int i=0; i<26; i++)
    cout << vett[i] << " lettere " << char(i+97) << endl;
  cout<<endl;
}

//Scrivete il vostro codice al di sotto di questo commento
int* calcola_frequenza (char parola[]){
   int *frequenza=new int[26];
   int i;

   for(i=0;i<26;i++)
     frequenza[i]=0;

   i=0;
   while (parola[i]!='\0' && i<dim) {
     frequenza[parola[i]-97]++;
     i++;
   }
   return frequenza;
}
```

2 Dato un file di testo contenente dati anagrafici relativi ad una persona nel seguente formato:

```
cognome nome citta_nascita data_nascita
```

in cui ogni campo indicato (`cognome`, `nome`, `citta_nascita`) può essere considerato come formato da una sola parola senza spazi bianchi o caratteri speciali e la data di nascita è scritta nel seguente formato: 4 cifre per l'anno, 2 cifre per il mese e 2 cifre per il giorno (quindi 31 marzo 2009 diventa 20090331).

Scrivere un programma nel file `esercizio2.cc` che, presi come argomento del `main` il nome di un file contenente questi dati anagrafici ordinati per data di nascita in modo crescente, richieda da tastiera un mese ed un anno e calcoli quante persone sono nate nell'anno specificato e nel mese ed anno specificato.

Dato ad esempio come file di input il file `anagrafica_studenti` contenente i seguenti dati:

```
BIANCHI STEFANO TORINO 19810131
CASIRAGHI UMBERTO PADOVA 19810202
ROSSI PAOLO GENOVA 19810228
AMADORI GIORGIO ROMA 19820715
FILIPPI ALESSANDRO ANCONA 19850101
TONONI ROBERTO MESSINA 19850923
CAPUTO SERGIO TRENTO 19861031
MODENA ALESSANDRO VENEZIA 19880511
BERTI LUIGI MILANO 19881012
```

se l'eseguibile è `a.out`, il comando

```
./a.out anagrafica_studenti
```

stamperà a video le seguenti informazioni:

```
Inserire il mese per cui ricercare il numero di nati:  2
Inserire l'anno per cui ricercare il numero di nati:  1981
Il numero di nati nell'anno 1981 e' 3
Il numero di nati nell'anno 1981 e nel mese 2 e' 2
```

NOTA 1:
visto che il file dati in input è ordinato il porgramma dovrà terminare il prima possibile

NOTA 2:
vista la codifica utilizzata per memorizzare la data di nascita delle persone, si suggerisce di estrarre i valori di mese e anno di nascita utilizzando la seguente forumula:

```
anno = <numero acquisto da file> / 10000
mese = (<numero acquisto da file> - anno * 10000) / 100
```

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

2 esercizio2.cc

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

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

  fstream my_in;
  char tmp[20];
  long int data;
  int mese_target, anno_target, mese, anno, numero_anno=0, numero_anno_mese=0, fine=0;

  if (argc != 2) {
    cout << "Usage: ./a.out <file_anagrafica> <file_frequenza>\n";
    exit(0);
  }

  cout << "Inserire il mese per cui ricercare il numero di nati: ";
  cin >> mese_target;

  cout << "Inserire l'anno per cui ricercare il numero di nati: ";
  cin >> anno_target;

  my_in.open(argv[1],ios::in);
  my_in >> tmp; //cognome
  while (!my_in.eof()&&(!fine)) {
    my_in >> tmp; //nome
    my_in >> tmp; //luogo
    my_in >> tmp; //data

    data = atoi(tmp);
    anno = data / 10000;
    mese = (data - (anno * 10000)) / 100;

    if (anno==anno_target)
      numero_anno++;
    if ((anno==anno_target)&&(mese==mese_target))
    numero_anno_mese++;
    if (data > ((anno_target*10000)+(mese_target*100)+31))
      fine=1;

    my_in >> tmp; //cognome
  }
  my_in.close();

  cout << "Il numero di nati nell'anno " << anno_target << " e' " << numero_anno <<endl;
  cout << "Il numero di nati nell'anno " << anno_target << " e nel mese " << mese_target << " e

  return(0);
}
```

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 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, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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

3 esercizio3.cc

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

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

void init(queue &q, int maxdim)
{
    q.tail = q.head = 0;
    q.dim = maxdim;
    q.elem = new float[maxdim];
}

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

static bool is_empty(const queue &q)
{
    return q.tail == q.head;
}

static bool is_full(const queue &q)
{
    return next(q.tail, q.dim) == q.head;
}

bool enqueue(queue &q, float n)
{
    if (is_full(q)) {
        return false;
    }
    q.elem[q.tail] = n;
    q.tail = next(q.tail, q.dim);
    return true;
}

bool dequeue(queue &q, float &out)
{
    if (is_empty(q)) {
        return false;
    }
    out = q.elem[q.head];
    q.head = next(q.head, q.dim);
    return true;
}

void print(const queue &q)
```

```
{
    for (int i = q.head; i != q.tail; i = next(i, q.dim)) {
        cout << q.elem[i] << " ";
    }
    cout << endl;
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menu per gestire una pila di `double`. 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` elementi;
- `deinit` liberi la memoria utilizzata dalla pila;
- `push` inserisca l'elemento passato come parametro nella pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `pop` tolga l'elemento in testa alla pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `top` legga l'elemento 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;
- `stampa` stampi a video il contenuto della pila, dall'elemento più vecchio al più recente.

La pila deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

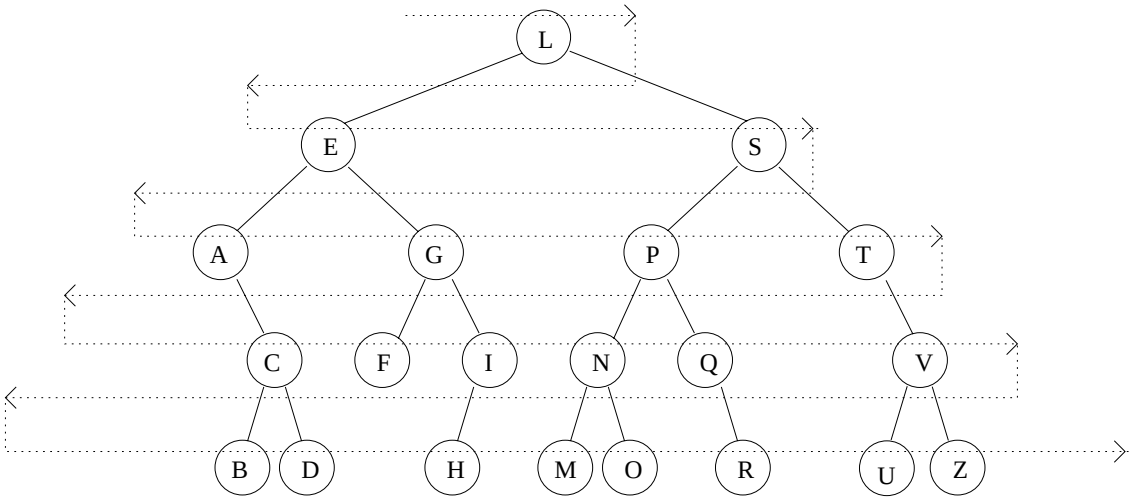
VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 10).

3 esercizio3.cc

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

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

static bool is_full(const stack &s)
{
    return s.index == s.dim;
}

void init(stack &s, int maxdim)
{
    s.index = 0;
    s.dim = maxdim;
    s.elem = new double[maxdim];
}

void deinit(stack &s)
{
    delete[] s.elem;
}

bool top(const stack &s, double &out)
{
    if (is_empty(s)) {
        return false;
    } else {
        out = s.elem[s.index-1];
        return true;
    }
}

bool push(stack &s, double n)
{
    if (is_full(s)) {
        return false;
    } else {
        s.elem[s.index++] = n;
        return true;
    }
}

bool pop(stack &s)
{
    if (is_empty(s)) {
        return false;
    } else {
        s.index--;
```

```
        return true;
    }
}

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

4 Per stampa breadth-first di un albero si intende la stampa degli elementi dell'albero partendo dalla radice e proseguendo per livelli di profondità crescenti, da sinistra a destra.



Ad esempio. la sequenza stampata in modalità breadth-first nella figura precedente è

L E S A G P T C F I N Q V B D H M O R U Z

Data la libreria `tree.h` per la gestione di alberi di ricerca binaria, si realizzi all'interno del file `esercizio4.cc` la funzione NON-RICORSIVA `stampa_bfs` che stampa un'albero in modo breadth-first.
NOTE:

- la funzione non deve essere ricorsiva;
- deve richiedere un numero di operazioni proporzionale alla dimensione dell'albero.

SUGGERIMENTO:
si usi la coda di alberi, disponibile nella libreria `queue.h`.

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

4 esercizio4.cc

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

void print_bfs(const tree & t);

int main()
{
  char option,val;
  tree t, tmp;
  retval res;
  init(t);
  do {
    cout << "\nOperazioni possibili:\n"
         << "Inserimento (i)\n"
         << "Ricerca (r)\n"
         << "Stampa ordinata (s)\n"
         << "Stampa indentata (e)\n"
         << "Stampa depth-first-search (d)\n"
         << "Fine (f)\n";
    cin >> option;
    switch (option) {
    case 'i':
      cout << "Val? : ";
      cin >> val;
      res = insert(t,val);
      if (res == FAIL)
        cout << "spazio insufficiente!\n";
      break;
    case 'r':
      cout << "Val? : ";
      cin >> val;
      tmp=cerca(t,val);
      if (!nullp(tmp))
        cout << "Valore trovato!: " << val << endl;
      else
        cout << "Valore non trovato!\n";
      break;
    case 's':
      cout << "Stampa ordinata:\n";
      print_ordered(t);
      break;
    case 'e':
      cout << "Stampa indentata:\n";
      print_indented(t);
      break;
    case 'd':
      cout << "Stampa breadth-first-search:\n";
      print_bfs(t);
      break;
    case 'f':
```

```
      break;
    default:
      cout << "Optione errata\n";
    }
  } while (option != 'f');
}


void print_bfs(const tree & t) {
  if (!nullp(t)) {
    enqueue(t);
    while (!queue_emptyp()) {
      tree t1;
      dequeue(t1);
      cout << t1->item << endl;
      if (t1->left!=NULL)
enqueue(t1->left);
      if (t1->right!=NULL)
enqueue(t1->right);
    }
  }
}
```