Primo Appelli di Programmazione I

30 Gennaio 2007
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 della Provetta

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

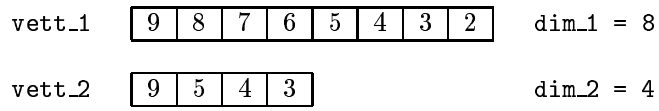
- Non è consentito l'uso di alcun libro di testo o fotocopia. In caso lo studente necessitasse di carta (?), gli/le verranno forniti fogli di carta bianca su richiesta, che dovranno essere riconsegnati a fine prova. È consentito l'uso di una penna. Non è consentito l'uso di alcuno strumento calcolatore.
- 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.

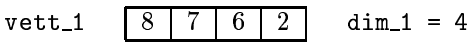
NOTA IMPORTANTE: DURANTE LA PROVA PER OGNI STUDENTE VERRÀ ATTIVATO UN TRACCIATORE SOFTWARE CHE REGISTRERÀ TUTTE LE OPERAZIONI ESEGUITE (ANCHE ALL'INTERNO DELL'EDITOR!!). L'ANNULLAMENTO DELLA PROVA DI UNO STUDENTE POTRÀ AVVENIRE ANCHE IN UN SECONDO MOMENTO, SE L'ANALISI DELLE TRACCE SOFTWARE RIVELASSERO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

1 Dato il file `esercizio1.cc`, scrivere una definizione **efficiente** della funzione `differenza` che, presi in input gli array di interi `vett_1` e `vett_2`, entrambi ordinati in modo decrescente, e le rispettive dimensioni `dim_1` e `dim_2`, sia in grado di restituire un nuovo array, ordinato in modo descrescente, contenente gli elementi presenti in `vett_1` ma non in `vett_2`, NULL altrimenti. Per semplicità, supporre che in ciascuno dei due array iniziali non ci siano elmenti duplicati.

Consideriamo ad esempio i seguenti array di interi:



In questo caso la funzione `differenza` deve restituire il vettore:

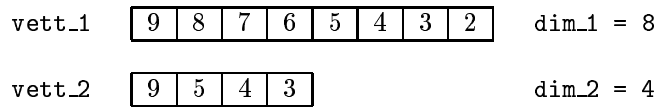


NOTA: quanti passi deve fare la vostra funzione se `vett_2` contiene un milione di elementi? Non è ritenuta efficiente una procedura che richiede un numero di passi dell'ordine di dim_1 $\times$ dim_2 o superiore.

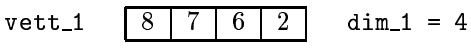
VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 Dato il file `esercizio1.cc`, scrivere una definizione **efficiente** della funzione `arraydiff` che, presi in input gli array di interi `vett_1` e `vett_2`, entrambi ordinati in modo decrescente, e le rispettive dimensioni `dim_1` e `dim_2`, sia in grado di restituire un nuovo array, ordinato in modo descrescente, contenente gli elementi presenti in `vett_1` ma non in `vett_2`, NULL altrimenti. Per semplicità, supporre che in ciascuno dei due array iniziali non ci siano elmenti duplicati.

Consideriamo ad esempio i seguenti array di interi:



In questo caso la funzione `arraydiff` deve restituire il vettore:



NOTA: quanti passi deve fare la vostra funzione se `vett_2` contiene un milione di elementi? Non è ritenuta efficiente una procedura che richiede un numero di passi dell'ordine di dim_1 $\times$ dim_2 o superiore.

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

1 **esercizio1.cc**

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

void leggi_array (int v[], int dim);
void stampa_array (int v[], int dim);
int* differenza (int vett_1[], int dim_1, int vett_2[], int dim_2, int &dim_3);

int main()
{
  const int MAX_DIM = 100;
  int vett_1[MAX_DIM], vett_2[MAX_DIM];
  int dim_1, dim_2;
  int *vett_3;
  int dim_3;

  cout << "Inserisci la dimensione di vett_1: "; cin >> dim_1;
  cout << "Inserisci i numeri dell'array vett_1: \n";
  leggi_array (vett_1, dim_1);

  cout << "Inserisci la dimensione di vett_2: "; cin >> dim_2;
  cout << "Inserisci i numeri dell'array vett_2: \n";
  leggi_array (vett_2, dim_2);

  vett_3 = differenza (vett_1, dim_1, vett_2, dim_2, dim_3);
  if (vett_3==NULL)
    cout << "Non e' stato trovato alcun numero presente in vett_1 ma non in vett_2!" << endl
  else {
    cout << "I numeri presenti in ";
    stampa_array (vett_1, dim_1);
    cout << " ma non in ";
    stampa_array (vett_2, dim_2);
    cout << " sono ";
    stampa_array (vett_3, dim_3);
      cout << "\n";
  }
}

void leggi_array (int vett[], int dim)
{
  int i;
  for(i=0; i<dim; i++)
    {
      cout << " numero " << i << ": ";
      cin >> vett[i];
    }
}

void stampa_array (int vett[], int dim)
{
  int i;
  for(i=0; i<dim; i++)
    {
```

```
        cout << vett[i] << " ";
    }
}

// Scrivi qui sotto il tuo codice

int* differenza (int vett_1[], int dim_1, int vett_2[], int dim_2, int &dim_3)
{
    int *result = new int[dim_1];
    dim_3 = 0;
    int i = 0, j = 0;
    while (i < dim_1 && j < dim_2) {
        if (vett_1[i] > vett_2[j]) {
            // siccome gli array sono ordinati, se l'elemento in posizione i
            // di vett_1 e' maggiore di quello in posizione j di vett_2,
            // sicuramente non compare in vett_2
            result[dim_3] = vett_1[i];
            ++dim_3;
            ++i;
        } else if (vett_1[i] == vett_2[j]) {
            ++i;
            ++j;
        } else { // vett_1[i] < vett_2[j]
            ++j;
        }
    }
    // controllo ulteriore, per gestire il caso in cui gli ultimi elementi di
    // vett_1 non siano in vett_2
    while (i < dim_1) {
        result[dim_3] = vett_1[i];
        ++dim_3;
        ++i;
    }

    if (dim_3 == 0) {
        delete [] result;
        return NULL;
    } else {
        return result;
    }
}
```

2 Scrivere un programma che, presi come argomenti del `main` due file copi il contenuto delle sole righe pari del secondo file alla fine del primo.

Esempio: se l'eseguibile è `a.out`, il comando
`./a.out testo1 testo2`
appenderà le righe pari di `testo2` in fondo a `testo1`.

NOTE:

- scrivere il codice relativo a questo esercizio in un file chiamato `esercizio2.cc` salvato nella directory `due`.
- per semplicità, assumere che nessuna riga sia più lunga di 1000 caratteri.

CONSIGLIO UTILE: per leggere dal file `myin` usare:

```
myin.getline(char* buffer, unsigned int num);
```

includendo la libreria `<fstream>`. `getline` legge in `buffer` caratteri da `myin` finché non incontra un terminatore di riga, o finchè il numero dei caratteri letti è maggiore o uguale a `num`. Il valore ritornato è `false` quando si è raggiunta la fine del file, `true` altrimenti.

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

2 Scrivere un programma che, presi come argomenti del `main` due file copi il contenuto delle sole righe dispari del secondo file alla fine del primo.

Esempio: se l'eseguibile è `a.out`, il comando
`./a.out testo1 testo2`
appenderà le righe dispari di `testo2` in fondo a `testo1`.

NOTE:

- scrivere il codice relativo a questo esercizio in un file chiamato `esercizio2.cc` salvato nella directory `due`.
- per semplicità, assumere che nessuna riga sia più lunga di 1000 caratteri.

CONSIGLIO UTILE: per leggere dal file `myin` usare:

```
myin.getline(char* buffer, unsigned int num);
```

includendo la libreria `<fstream>`. `getline` legge in `buffer` caratteri da `myin` finché non incontra un terminatore di riga, o finchè il numero dei caratteri letti è maggiore o uguale a `num`. Il valore ritornato è `false` quando si è raggiunta la fine del file, `true` altrimenti.

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

2 **esercizio2.cc**

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

int main (int argc, char * argv[])
{
  fstream myin,myout;
  char riga[1001];

  if (argc!=3) {
    cout << "Usage: ./a.out <destinazione> <sorgente>\n";
    exit(0);
  }

  myin.open(argv[2],ios::in);
  myout.open(argv[1],ios::out|ios::app);

  int riga_corrente = 0;

  while (myin.getline(riga, 1001)) {
      ++riga_corrente;
      if (riga_corrente % 2 == 0) {
          myout << riga << endl;
      }
  }
  myin.close();
  myout.close();
}
```

3 Implementare un albero di ricerca binaria, i cui nodi contengano le informazioni *nome e cognome*, con le seguenti funzionalità:

- inizializzazione della struttura dati albero;
- controllo se un albero è vuoto;
- inserimento di un elemento con il seguente ordine alfabetico crescente per cognome: i cognomi più bassi ($a < b < c < \ldots$) a sinistra; per semplicità, si suppone che non vengano mai inseriti cognomi già presenti nell'albero;
- ricerca di un elemento per cognome;
- stampa dell'albero in ordine alfabetico. Esempio: l'albero seguente

$$\textit{Roberto Sebastiani}$$

$$\textit{Alberto Griggio} \qquad\qquad \textit{Alessandro Tomasi}$$

deve essere stampato come:
Alberto Griggio
Roberto Sebastiani
Alessandro Tomasi.

Scrivere nel file `albero.cc` la definizione delle funzioni dichiarate nello header file `albero.h`. Nel file `albero_main.cc` è definita la funzione `main` che contiene un menu per gestire l'albero.

CONSIGLI UTILI: Per copiare e confrontare stringhe usare:

```
char *strcpy(char *dest, const char *src);
int strcmp(const char *s1, const char *s2);
int strlen(const char * string );
```

includendo la libreria `<string.h>`.

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

3 Implementare un albero di ricerca binaria, i cui nodi contengano le informazioni *nome e cognome*, con le seguenti funzionalità:

- inizializzazione della struttura dati albero;
- controllo se un albero è vuoto;
- inserimento di un elemento con il seguente ordine alfabetico crescente per cognome: i cognomi più bassi ($a < b < c < \ldots$) a sinistra; per semplicità, si suppone che non vengano mai inseriti cognomi già presenti nell'albero;
- ricerca di un elemento per cognome;
- stampa dell'albero in ordine alfabetico. Esempio: l'albero seguente

  Roberto Sebastiani

  Alberto Griggio *Alessandro Tomasi*

  deve essere stampato come:
  Alberto Griggio
  Roberto Sebastiani
  Alessandro Tomasi.

Scrivere nel file `albero.cc` la definizione delle funzioni dichiarate nello header file `albero.h`. Nel file `albero_main.cc` è definita la funzione `main` che contiene un menu per gestire l'albero.

CONSIGLI UTILI: Per copiare e confrontare stringhe usare:

```
char *strcpy(char *dest, const char *src);
int strcmp(const char *s1, const char *s2);
int strlen(const char * string );
```

includendo la libreria `<string.h>`.

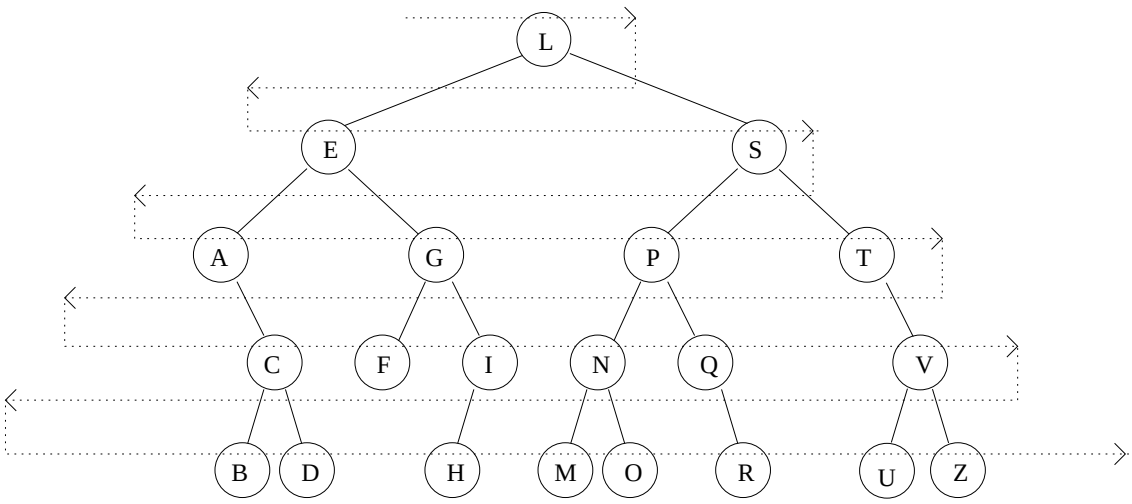
VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

3 esercizio3.cc

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

struct elem{
  char nome[20];
  char cognome[20];
  elem * ds ;
  elem * sn ;
};

//static elem* tree;

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

retval vuoto(Tree &tree)
{
  if (tree==NULL) return OK;
  else return FAIL;
}

void stampa_elem(elem * t)
{
  cout << "Nome: " << t->nome << endl;
  cout << "Cognome: " << t->cognome << endl;
}


elem * ins(char* nome, char* cognome, elem *t)
{
  if (t == NULL) {
    elem *app = new elem;
    strcpy(app->nome,nome) ;
    strcpy(app->cognome,cognome);
    app->ds = NULL;
    app->sn = NULL;
    return app;
  } else {
      int result = strcmp(cognome, t->cognome);
      if (result == 0) {
          return t; // impossibile inserire, non faccio niente
      } else if (result < 0) {
          t->sn = ins(nome, cognome, t->sn);
      } else {
          t->ds = ins(nome, cognome, t->ds);
      }
      return t;
  }
}
```

```
void inserimento(Tree &tree, char *nome,char *cognome) {
  tree = ins(nome,cognome,tree);
}



retval ric (elem* t,char * cognome) {
  if (t==NULL)
    return FAIL;
  else
    if(strcmp(cognome,t->cognome)==0)
      {stampa_elem(t); return OK;}
    else
      if (strcmp(cognome,t->cognome) < 0)
        return ric(t->sn,cognome);
      else
        return ric(t->ds,cognome);
}

retval ricerca (Tree &tree, char *cognome)
{
  return ric(tree,cognome);
}

void stampa(elem * tree)
{
  if (tree!=NULL)  {
    stampa(tree->sn);
    stampa_elem(tree);
    stampa(tree->ds);
  }
}

void stampa_ordinata(Tree &tree) {
  stampa(tree);
}
```

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);
    }
  }
}
```