Quinto Appello di Programmazione I

07 Settembre 2011
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 un programma, nel file `esercizio1.cc`, che, presi come argomenti del `main` i nomi di due file, copi il primo file nel secondo correggendone la sintassi e generando in tal modo un testo corretto. Affinché un testo possa essere considerato corretto, devono essere verificate le seguenti regole:

(a) la prima parola del testo deve iniziare con una lettera maiuscola;

(b) tutte le parole che seguono i seguenti caratteri: ".", "?" e "!", devono iniziare con una lettera maiuscola.

Se ad esempio l'eseguibile è `a.out`, il comando

```
./a.out testo testocorretto
```

creerà un nuovo file di nome `testocorretto` e vi copierà il contenuto del file dato `testo`, modificando le parole quando queste non verificano le regole descritte sopra. Nelle figure 1 e 2 un esempio di file `testo` e `testocorretto`.

filastrocca delle parole:
Fatevi avanti! chi ne vuole?
di parole ho la testa piena,
con dentro la "luna" e la "balena".
ci sono parole per gli amici:
Buon giorno, Buon anno, Siate felici!
parole belle e parole buone;
parole per ogni sorta di persone.
di G. Rodari.

Figura 1: `testo`

Filastrocca delle parole:
Fatevi avanti! Chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
Buon giorno, Buon anno, Siate felici!
Parole belle e parole buone;
parole per ogni sorta di persone.
Di G. Rodari.

Figura 2: `testocorretto`

NOTA 1: Per semplicità si assuma che il testo contenuto nel primo file sia su un'unica riga ed inizi con un carattere alfabetico, non contenga "..." e che ".", "?" e "!" siano sempre preceduti da una parola e seguiti da uno spazio.

NOTA 2: il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). Per realizzare la conversione da caratteri minuscoli in maiuscoli, è vietato l'uso di tabelle o di 26 `if` o `switch-case`, uno per ogni carattere.

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

1 **esercizio1.cc**

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

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

  fstream my_in, my_out;
  char tmp[30];

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

  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  my_in >> tmp;
  if(!(tmp[0] >= 'A' && tmp[0] <= 'Z'))
    tmp[0] = tmp[0] + ('A'-'a');

  while(!my_in.eof()){
    my_out << tmp << " ";
    if(tmp[strlen(tmp)-1] == '.' || tmp[strlen(tmp)-1] == '?' || tmp[strlen(tmp)-1] == '!') {
      my_in >> tmp;
      if(!(tmp[0] >= 'A' && tmp[0] <= 'Z'))
        tmp[0] = tmp[0] + ('A'-'a');
    } else my_in >> tmp;
  }

  my_in.close();
  my_out.close();
  return(0);
}
```

1 Scrivere un programma, nel file `esercizio1.cc`, che, presi come argomenti del `main` i nomi di due file, copi il primo file nel secondo correggendone la sintassi e generando in tal modo un testo corretto. Affinché un testo possa essere considerato corretto, devono essere verificate le seguenti regole:

(a) la prima parola del testo deve iniziare con una lettera maiuscola;

(b) tutte le parole che seguono i seguenti caratteri: ",", ";" e ":", devono iniziare con una lettera minuscola.

Se ad esempio l'eseguibile è `a.out`, il comando

```
./a.out testo testocorretto
```

creerà un nuovo file di nome `testocorretto` e vi copierà il contenuto del file dato `testo`, modificando le parole quando queste non verificano le regole descritte sopra. Nelle figure 1 e 2 un esempio di file `testo` e `testocorretto`.

filastrocca delle parole:
Fatevi avanti! chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
Buon giorno, buon anno, Siate felici!
Parole belle e parole buone;
Parole per ogni sorta di persone.
Di G. Rodari.

Figura 3: `testo`

Filastrocca delle parole:
fatevi avanti! chi ne vuole?
Di parole ho la testa piena,
con dentro la "luna" e la "balena".
Ci sono parole per gli amici:
buon giorno, buon anno, siate felici!
Parole belle e parole buone;
parole per ogni sorta di persone.
Di G. Rodari.

Figura 4: `testocorretto`

NOTA 1: Per semplicità si assuma che il testo contenuto nel primo file sia su un'unica riga, inizi con un carattere alfabetico e che ",", ";" e ":" siano sempre preceduti da una parola e seguiti da uno spazio.

NOTA 2: il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). Per realizzare la conversione da caratteri minuscoli in maiuscoli o viceversa, è vietato l'uso di tabelle o di 26 `if` o `switch-case`, uno per ogni carattere.

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

1 **esercizio1.cc**

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

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

  fstream my_in, my_out;
  char tmp[30];

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

  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  my_in >> tmp;
  if(tmp[0] >= 'a' && tmp[0] <= 'z')
    tmp[0] = tmp[0] + ('A'-'a');

  while(!my_in.eof()){
    my_out << tmp << " ";
    if(tmp[strlen(tmp)-1] == ',' || tmp[strlen(tmp)-1] == ';' || tmp[strlen(tmp)-1] == ':') {
      my_in >> tmp;
      if(!(tmp[0] >= 'a' && tmp[0] <= 'z'))
        tmp[0] = tmp[0] - ('A'-'a');
    } else my_in >> tmp;
  }

  my_in.close();
  my_out.close();
  return(0);
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione `rimuovi_carattere` che, presi come parametri una stringa `parola` ed un carattere `d`, rimuova da `parola` ogni occorrenza di `d`. La funzione deve inoltre restituire, tramite un terzo parametro formale, il numero delle occorrenze del carattere `d` rimosse. Per esempio, data la stringa ``arrivederci'' ed il carattere `'r'`, la funzione `rimuovi_carattere` produce la stringa ``aivedeci'' e restituisce l'intero `3`.

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

2 **esercizio2.cc**

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

// Dichiarare qui sotto la funzione rimuovi_carattere

void rimuovi_carattere(char parola[], char d, int& num);

int main () {
   char stringa[100];
   char carattere;
   int occorrenze;

   cout << "Inserisci parola: ";
   cin >> stringa;

   cout << "Inserisci carattere: ";
   cin >> carattere;

   rimuovi_carattere(stringa, carattere, occorrenze);

   cout << "Risultato: " << stringa;
   cout << " (tolte " << occorrenze << " occorrenze di " << carattere << ")\n";

   return 0;
}

// Definire qui sotto la funzione rimuovi_carattere

void rimuovi_carattere(char parola[], char d, int& num) {

   num = 0;
   int i = 0, j = 0;
   while (parola[i] != '\0') {
      if (parola[i] != d) {
         parola[j] = parola[i];
         j++;
      } else {
         num++;
      }
      i++;
   }
   parola[j]='\0';
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione `rimuovi_carattere` che, presi come parametri una stringa `parola` ed un carattere `d`, rimuova da `parola` ogni occorrenza di `d`. La funzione deve inoltre restituire, tramite un terzo parametro formale, la lunghezza della stringa ottenuta eliminando le occorrenze di `d` da `parola`. Per esempio, data la stringa ``buongiorno'' ed il carattere `'n'`, la funzione `rimuovi_carattere` produce la stringa ``buogioro'' e restituisce l'intero `3`.

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

2 **esercizio2.cc**

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

// Dichiarare qui sotto la funzione rimuovi_carattere

void rimuovi_carattere(char parola[], char d, int& num);

int main () {
   char stringa[100];
   char carattere;
   int occorrenze;

   cout << "Inserisci parola: ";
   cin >> stringa;

   cout << "Inserisci carattere: ";
   cin >> carattere;

   rimuovi_carattere(stringa, carattere, occorrenze);

   cout << "Risultato: " << stringa;
   cout << " (lunghezza " << occorrenze << ")\n";

   return 0;
}

// Definire qui sotto la funzione rimuovi_carattere

void rimuovi_carattere(char parola[], char d, int& num) {

   num = 0;
   int i = 0, j = 0;
   while (parola[i] != '\0') {
      if (parola[i] != d) {
         parola[j] = parola[i];
         j++;
         num++;
      }
      i++;
   }
   parola[j]='\0';
}
```

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);
    }
}
```

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 **decrescente**. Esempio: l'inserimento dei seguenti valori:

  ```
  e  a  u
  ```

  deve produrre il seguente albero:

  ```
      e
    u   a
  ```

- `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 **decrescente**. Esempio: l'albero qui sopra deve essere stampato come:

  ```
  u  e  a
  ```

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
```

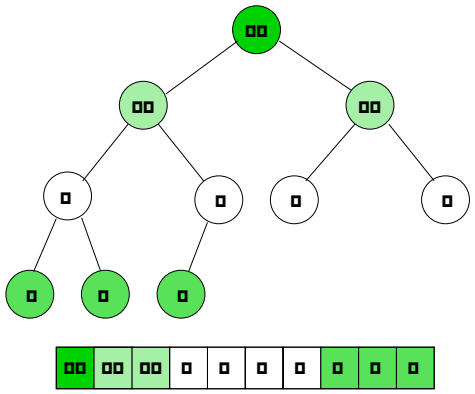
3 **soluzione_A32.cc**

```
#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 maggiori di t->val (cioe' quelli a sx)
        print(t->left);
        // poi stampo t->val
        cout << t-> val << ' ';
        // poi stampo gli elementi minori (cioe' quelli a dx)
        print(t->right);
    }
}
```

4 Un binary heap è una struttura dati che consiste in un albero binario quasi completo (i.e. riempito completamente su tutti i livelli tranne, eventualmente, l'ultimo che è riempito da sinistra a destra) dove il valore di un nodo figlio è minore o uguale a quello del nodo padre. Lo heap può essere efficentemente memorizzato con un array `A`, dove `A[0]` è il nodo radice e, se i è l'indice di un nodo, l'idice del padre `parent(i)`, del figlio sinistro `left(i)` e del figlio destro `right(i)` possono essere rispettivamente calcolati con $\lceil (i-1)/2 \rceil$, $2i+1$, $2i+2$.

La figura sottostante mostra un binary heap che contiene 10 elementi e l'array utilizzato per memorizzarlo.



Sono dati:

- il file `binaryheap.h`, contenente la definizione delle strutture dati e gli header delle funzioni
- il file binario `binaryheap.o`, contenente le funzioni `stampa_heap` e `rimuovi_elemento` usate nel main
- il file `esercizio4.cc`, contenente la funzione `main`.

Definire in `esercizio4.cc` la funzione `inserisci_elemento` che inserisce un elmento nello heap.

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

4 esercizio4.cc

```
using namespace std;
#include "binaryheap.h"

int main () {
   char res;
   binaryheap h;
   int n;

   build_heap(h,20);
   do {
      cout << "\nOperazioni possibili:\n"
           << "Inserisci (i): inserisci un elemento\n"
           << "Estrai (e): estrai l'elemento massimo\n"
           << "Stampa (s): stampa heap\n"
           << "Fine (f)\n";
      cout << "\nScegli operazione: ";
      cin >> res;
      switch (res) {
      case 'i':
         cin >> n;
         if (inserisci_elemento(h,n)==FAIL) {
            cout << "Heap pieno!\n";;
         }
         break;
      case 'e':
         if (estrai_massimo(h,n)==OK) {
            cout << n << endl;
         } else {
            cout << "Heap vuota\n";
         }
         break;
      case 's':
         stampa_heap(h);
         break;
      case 'f':
         break;
      default:
         cout << "Valore errato!\n";
      }
   } while (res != 'f');
}

// Definire qui sotto la funzione inserisci_elemento

static boolean pieno(binaryheap & h) {
   if (h.size == DIM) {
      return TRUE;
   }
   return FALSE;
}

retval inserisci_elemento(binaryheap & h, int elem) {
```

```
   if (pieno(h)==TRUE) {
      return FAIL;
   }
   h.size++;
int i = h.size;

while(i > 1 && h.array[i/2-1] < elem) {
    h.array[i-1] = h.array[i/2-1];
    i = i/2;
}
h.array[i-1] = elem;
   return OK;
}
```