Secondo Appello di Programmazione I

16 Febbraio 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 nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file di testo, legga il contenuto di tali file e generi un terzo file contenente solo le righe presenti in entrambi i file.

In particolare, i file indicati come argomento conterrano una serie di valori **interi** e spazi bianchi organizzati su più righe **ordinati** in modo alfabetico crescente riga per riga.

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

```
2 5 0 1
35 0 123
4 5 6
7 81 99
```

e il file `secondo` ha il seguente contenuto:

```
000
2 5 0 1
4 5 6
7 81 99
```

allora il comando:
`./a.out primo secondo terzo`
genera un file chimato `terzo` con il seguente contenuto:

```
2 5 0 1
4 5 6
7 81 99
```

NOTA 1: è possibile usare dalla libreria `<fstream>` il metodo `<nome file>.eof()` per riconoscere la fine di un file e il metodo `<nome file>.getline()` per leggere una riga intera di un file di testo.

NOTA 2: Si ricorda che i file stream utilizzati all'interno del codice devono essere correttamente chiusi prima della fine del programma.

NOTA 3: dato che i file di input sono entrambi ordinati, il programma dovrà terminare il prima possibile.

NOTA 4: per semplicità supporre che ogni riga contenga al massimo 255 caratteri.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

int main (int argc, char* argv[])
{
  fstream f_primo, f_secondo, f_terzo;
  char tmp1[256], tmp2[256];

  if (argc != 4)
  {
    cout << "Eseguire: " << argv[0] << " <nome-primo-file> <nome-secondo-file> <nome-terzo-file
    return 1;
  }

  f_primo.open(argv[1], ios::in);
  f_secondo.open(argv[2], ios::in);
  f_terzo.open(argv[3], ios::out);


  f_primo.getline(tmp1, 256);  //lettura prima riga dal primo file
  f_secondo.getline(tmp2, 256);  //lettura prima riga dal secondo file

  while ((!f_primo.eof()) && (!f_secondo.eof())) {
    if (strcmp(tmp1,tmp2)==0)
      {
        f_terzo << tmp1 << endl;
        f_primo.getline(tmp1, 256);
        f_secondo.getline(tmp2, 256);
      }
    if (strcmp(tmp1,tmp2)>0)
      {
        f_secondo.getline(tmp2, 256);
      }
    if (strcmp(tmp1,tmp2)<0)
      {
        f_primo.getline(tmp1, 256);
      }
  }

  f_primo.close();
  f_secondo.close();
  f_terzo.close();

  return 0;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file di testo, legga il contenuto di tali file e generi un terzo file contenente solo le righe presenti in entrambi i file.

In particolare, i file indicati come argomento conterrano una serie di valori **interi** e spazi bianchi organizzati su più righe **ordinati** in modo alfabetico decrescente riga per riga.

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

```
7 81 99
4 5 6
35 0 123
2 5 0 1
```

e il file `secondo` ha il seguente contenuto:

```
7 81 99
4 5 6
2 5 0 1
000
```

allora il comando:
`./a.out primo secondo terzo`
genera un file chimato `terzo` con il seguente contenuto:

```
7 81 99
4 5 6
2 5 0 1
```

NOTA 1: è possibile usare dalla libreria `<fstream>` il metodo `<nome file>.eof()` per riconoscere la fine di un file e il metodo `<nome file>.getline()` per leggere una riga intera di un file di testo.

NOTA 2: Si ricorda che i file stream utilizzati all'interno del codice devono essere correttamente chiusi prima della fine del programma.

NOTA 3: dato che i file di input sono entrambi ordinati, il programma dovrà terminare il prima possibile.

NOTA 4: per semplicità supporre che ogni riga contenga al massimo 255 caratteri.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

int main (int argc, char* argv[])
{
  fstream f_primo, f_secondo, f_terzo;
  char tmp1[256], tmp2[256];

  if (argc != 4)
  {
    cout << "Eseguire: " << argv[0] << " <nome-primo-file> <nome-secondo-file> <nome-terzo-file
    return 1;
  }

  f_primo.open(argv[1], ios::in);
  f_secondo.open(argv[2], ios::in);
  f_terzo.open(argv[3], ios::out);


  f_primo.getline(tmp1, 256);  //lettura prima riga dal primo file
  f_secondo.getline(tmp2, 256);  //lettura prima riga dal secondo file

  while ((!f_primo.eof()) && (!f_secondo.eof())) {
    if (strcmp(tmp1,tmp2)==0)
      {
        f_terzo << tmp1 << endl;
        f_primo.getline(tmp1, 256);
        f_secondo.getline(tmp2, 256);
      }
    if (strcmp(tmp1,tmp2)<0)
      {
        f_secondo.getline(tmp2, 256);
      }
    if (strcmp(tmp1,tmp2)>0)
      {
        f_primo.getline(tmp1, 256);
      }
  }

  f_primo.close();
  f_secondo.close();
  f_terzo.close();

  return 0;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file di testo, legga il contenuto di tali file e generi un terzo file contenente solo le righe presenti in entrambi i file.

In particolare, i file indicati come argomento conterrano una serie di **caratteri alfabetici** e spazi bianchi organizzati su più righe **ordinati** in modo alfabetico crescente riga per riga.

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

```
b h aa
ef ggg
mno pq
www
```

e il file `secondo` ha il seguente contenuto:

```
aaa
b h aa
hh aa
mno pq
zzz
```

allora il comando:
`./a.out primo secondo terzo`
genera un file chimato `terzo` con il seguente contenuto:

```
b h aa
mno pq
```

NOTA 1: è possibile usare dalla libreria `<fstream>` il metodo `<nome file>.eof()` per riconoscere la fine di un file e il metodo `<nome file>.getline()` per leggere una riga intera di un file di testo.

NOTA 2: Si ricorda che i file stream utilizzati all'interno del codice devono essere correttamente chiusi prima della fine del programma.

NOTA 3: dato che i file di input sono entrambi ordinati, il programma dovrà terminare il prima possibile.

NOTA 4: per semplicità supporre che ogni riga contenga al massimo 255 caratteri.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

int main (int argc, char* argv[])
{
  fstream f_primo, f_secondo, f_terzo;
  char tmp1[256], tmp2[256];

  if (argc != 4)
  {
    cout << "Eseguire: " << argv[0] << " <nome-primo-file> <nome-secondo-file> <nome-terzo-file
    return 1;
  }

  f_primo.open(argv[1], ios::in);
  f_secondo.open(argv[2], ios::in);
  f_terzo.open(argv[3], ios::out);


  f_primo.getline(tmp1, 256);  //lettura prima riga dal primo file
  f_secondo.getline(tmp2, 256);  //lettura prima riga dal secondo file

  while ((!f_primo.eof()) && (!f_secondo.eof())) {
    if (strcmp(tmp1,tmp2)==0)
      {
        f_terzo << tmp1 << endl;
        f_primo.getline(tmp1, 256);
        f_secondo.getline(tmp2, 256);
      }
    if (strcmp(tmp1,tmp2)>0)
      {
        f_secondo.getline(tmp2, 256);
      }
    if (strcmp(tmp1,tmp2)<0)
      {
        f_primo.getline(tmp1, 256);
      }
  }

  f_primo.close();
  f_secondo.close();
  f_terzo.close();

  return 0;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file di testo, legga il contenuto di tali file e generi un terzo file contenente solo le righe presenti in entrambi i file.

In particolare, i file indicati come argomento conterrano una serie di **caratteri alfabetici** e spazi bianchi organizzati su più righe **ordinati** in modo alfabetico decrescente riga per riga.

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

```
www
mno pq
ef ggg
b h aa
```

e il file `secondo` ha il seguente contenuto:

```
zzz
mno pq
hh aa
b h aa
aaa
```

allora il comando:
`./a.out primo secondo terzo`
genera un file chimato `terzo` con il seguente contenuto:

```
mno pq
b h aa
```

NOTA 1: è possibile usare dalla libreria `<fstream>` il metodo `<nome file>.eof()` per riconoscere la fine di un file e il metodo `<nome file>.getline()` per leggere una riga intera di un file di testo.

NOTA 2: Si ricorda che i file stream utilizzati all'interno del codice devono essere correttamente chiusi prima della fine del programma.

NOTA 3: dato che i file di input sono entrambi ordinati, il programma dovrà terminare il prima possibile.

NOTA 4: per semplicità supporre che ogni riga contenga al massimo 255 caratteri.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

int main (int argc, char* argv[])
{
  fstream f_primo, f_secondo, f_terzo;
  char tmp1[256], tmp2[256];

  if (argc != 4)
  {
    cout << "Eseguire: " << argv[0] << " <nome-primo-file> <nome-secondo-file> <nome-terzo-file
    return 1;
  }

  f_primo.open(argv[1], ios::in);
  f_secondo.open(argv[2], ios::in);
  f_terzo.open(argv[3], ios::out);


  f_primo.getline(tmp1, 256);  //lettura prima riga dal primo file
  f_secondo.getline(tmp2, 256);  //lettura prima riga dal secondo file

  while ((!f_primo.eof()) && (!f_secondo.eof())) {
    if (strcmp(tmp1,tmp2)==0)
      {
        f_terzo << tmp1 << endl;
        f_primo.getline(tmp1, 256);
        f_secondo.getline(tmp2, 256);
      }
    if (strcmp(tmp1,tmp2)<0)
      {
        f_secondo.getline(tmp2, 256);
      }
    if (strcmp(tmp1,tmp2)>0)
      {
        f_primo.getline(tmp1, 256);
      }
  }

  f_primo.close();
  f_secondo.close();
  f_terzo.close();

  return 0;
}
```

2 Scrivere nel file `esercizio2.cc` la definizione della funzione `pattern_matcher` che verifica (in modo iterativo) se un array di interi, chiamato `pattern`, è contenuto in un altro array di interi, chiamato `testo`. Per esempio, l'array $[9, 1, 7]$ è contenuto nell'array $[4, 5, 9, 1, 7, 3]$, mentre l'array $[5, 1]$ non lo è. In particolare, tale funzione ritorna `true` se `pattern` è contenuto in `testo`, `false` altrimenti.

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

2 **esercizio2.cc**

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

#define SIZE 20

void inizializza_array(int array[], int &dim);

bool pattern_matcher(int seq[], int n, int subseq[], int m);


int main() {

   int testo[SIZE];
   int pattern[SIZE];
   int n, m;

   cout << "Creazione del testo (<= " << SIZE <<")" << endl;
   inizializza_array(testo, n);

   cout << "Creazione del pattern (<= " << SIZE <<")" << endl;
   inizializza_array(pattern, m);

   if (m > n) {
      cout << "Pattern piu' lungo del testo!" << endl;
      return 0;
   }

   bool res = pattern_matcher(testo, n, pattern, m);

   if (res) {
      cout << "Il pattern occorre nel testo!";
   } else {
      cout << "Il pattern NON occorre nel testo!";
   }
   cout << endl;

   return 0;
}

void inizializza_array(int array[], int &dim) {
   cout << "Dimensione array: ";
   cin >> dim;
   cout << "Elementi: " << endl;
   for (int i=0; i<dim; i++) {
      cin >> array[i];
   }
}

//Inserire qui la definizione della funzione pattern_matcher

bool pattern_matcher(int seq[], int n, int subseq[], int m) {
   int i,j;
```

```
   for (i=0; i<=n-m; i++) {
      for(j=0; j<m && subseq[j] == seq[i+j]; j++);
      if (j==m) {
         return true;
      }
   }
   return false;
}
```

2 Scrivere nel file `esercizio2.cc` la definizione della funzione `is_substring` che verifica (in modo iterativo) se un array di float, chiamato `substring`, è contenuto in un altro array di caratteri, chiamato `text`. Per esempio, dati gli array $[9.3, 1, 7.8]$ e $[5.5, 1]$, il primo è contenuto nell'array $[4.1, 5.5, 9.3, 1, 7.8, 3]$ ma il secondo no. Inoltre, la funzione `is_substring` ritornerà `true` se `substring` è contenuto in `text`, `false` altrimenti.

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

2 **esercizio2.cc**

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

#define SIZE 20

void inizializza_array(float array[], int &dim);

bool is_substring(float seq[], int n, float subseq[], int m);


int main() {

   float text[SIZE];
   float substring[SIZE];
   int n, m;

   cout << "Creazione di text (<= " << SIZE <<")" << endl;
   inizializza_array(text, n);

   cout << "Creazione di substring (<= " << SIZE <<")" << endl;
   inizializza_array(substring, m);

   if (m > n) {
      cout << "Substring piu‘ lunga di text!" << endl;
      return 0;
   }

   bool res = is_substring(text, n, substring, m);

   if (res) {
      cout << "Substring occorre in text!";
   } else {
      cout << "Substring NON occorre in text!";
   }
   cout << endl;

   return 0;
}

void inizializza_array(float array[], int &dim) {
   cout << "Dimensione array: ";
   cin >> dim;
   cout << "Elementi: " << endl;
   for (int i=0; i<dim; i++) {
      cin >> array[i];
   }
}

//Inserire qui la definizione della funzione pattern_matcher

bool is_substring(float seq[], int n, float subseq[], int m) {
   int i,j;
```

```
   for (i=0; i<=n-m; i++) {
      for(j=0; j<m && subseq[j] == seq[i+j]; j++);
      if (j==m) {
         return true;
      }
   }
   return false;
}
```

2 Scrivere nel file `esercizio2.cc` la definizione della funzione `check_pattern` che verifica (in modo iterativo) se un array di long, chiamato `pattern`, è contenuto in un altro array di interi, chiamato `text`. Esempio: l'array $[1, 2]$ non è contenuto nell'array $[4, 1, 8, 2, 7, 3]$, mentre l'array $[8, 2, 7]$ lo è. Tale funzione dove ritornare `true` se `pattern` è contenuto in `text`, `false` altrimenti.

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

2 **esercizio2.cc**

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

#define SIZE 20

void inizializza_array(long array[], int &dim);

bool check_pattern(long seq[], int n, long subseq[], int m);


int main() {

   long text[SIZE];
   long pattern[SIZE];
   int n, m;

   cout << "Creazione del testo (<= " << SIZE <<")" << endl;
   inizializza_array(text, n);

   cout << "Creazione del pattern (<= " << SIZE <<")" << endl;
   inizializza_array(pattern, m);

   if (m > n) {
      cout << "Pattern piu' lungo di text!" << endl;
      return 0;
   }

   bool res = check_pattern(text, n, pattern, m);

   if (res) {
      cout << "Pattern occorre in text!";
   } else {
      cout << "Pattern NON occorre in text!";
   }
   cout << endl;

   return 0;
}

void inizializza_array(long array[], int &dim) {
   cout << "Dimensione array: ";
   cin >> dim;
   cout << "Elementi: " << endl;
   for (int i=0; i<dim; i++) {
      cin >> array[i];
   }
}

//Inserire qui la definizione della funzione check_pattern

bool check_pattern(long seq[], int n, long subseq[], int m) {
   int i,j;
```

```
   for (i=0; i<=n-m; i++) {
      for(j=0; j<m && subseq[j] == seq[i+j]; j++);
      if (j==m) {
         return true;
      }
   }
   return false;
}
```

2 Scrivere nel file `esercizio2.cc` la definizione della funzione `substring_matcher` che verifica (in modo iterativo) se un array di caratteri, chiamato `sottostringa`, è contenuto in un altro array di caratteri, chiamato `stringa`. Esempio: l'array $[s, v, b]$ è contenuto nell'array $[z, a, s, v, b, e]$, invece l'array $[z, s]$ non lo è. Si noti che tale funzione dovrà ritornare `true` se `sottostringa` è contenuto in `stringa`, `false` altrimenti.

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

2 **esercizio2.cc**

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

#define SIZE 20

void inizializza_array(char array[], int &dim);

bool substring_matcher(char seq[], int n, char subseq[], int m);


int main() {

   char sottostringa[SIZE];
   char stringa[SIZE];
   int n, m;

   cout << "Creazione della stringa (<= " << SIZE <<")" << endl;
   inizializza_array(stringa, n);

   cout << "Creazione della sottostringa (<= " << SIZE <<")" << endl;
   inizializza_array(sottostringa, m);

   if (m > n) {
      cout << "Sottostringa piu‘ lunga della sottostringa!" << endl;
      return 0;
   }

   bool res = substring_matcher(stringa, n, sottostringa, m);

   if (res) {
      cout << "La sottostringa occorre nella stringa!";
   } else {
      cout << "La sottostringa NON occorre nella stringa!";
   }
   cout << endl;

   return 0;
}

void inizializza_array(char array[], int &dim) {
   cout << "Dimensione array: ";
   cin >> dim;
   cout << "Elementi: " << endl;
   for (int i=0; i<dim; i++) {
      cin >> array[i];
   }
}

//Inserire qui la definizione della funzione pattern_matcher

bool substring_matcher(char seq[], int n, char subseq[], int m) {
   int i,j;
```

```
   for (i=0; i<=n-m; i++) {
      for(j=0; j<m && subseq[j] == seq[i+j]; j++);
      if (j==m) {
         return true;
      }
   }
   return false;
}
```

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_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 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);
    }
}
```

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
```

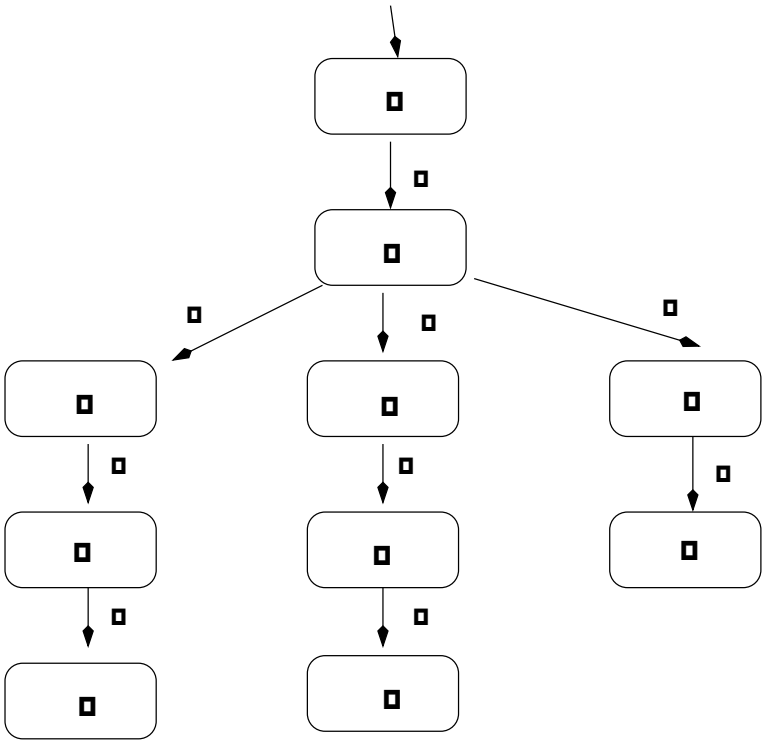
3 **soluzione_A31.cc**

```
#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 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 wordtree è una struttura dati che consente di memorizzare un gran numero di occorrenze di parole minuscole e recuperarle in modo estremamente efficiente.
Consiste in un albero di ricerca in cui ogni nodo ha 26 sottoalberi figli (uno per ogni carattere minuscolo) e contiene un intero che rappresenta il numero di occorrenze della parola rappresentata dal ramo di cui il nodo e' l'elemento terminale.

Ad esempio, la sequenza di parole `bidi bi bodi bi bum`, è rappresentata dal wordtree in figura.



Sono dati:

- il file `wordtree.h`, contenente la definizione delle strutture dati e gli header delle funzioni
- il file binario `wordtree.o`, contenente la funzione `stampa_wordtree` usata nel main
- il file `esercizio4.cc`, contenente la funzione main().

Definire in `esercizio4.cc` le funzioni `aggiungi_parola`, che aggiunge una nuova parola al wordtree, e `occorrenze_parola`, che resturuisce il numero di occorrenze di parola in wordtree.

Esempio: se il file `in` contiene il testo:

```
bimbo ba bi bo bu bimba
bi ba ba ba bumba
bidi bodi bim bum bam
```

il programma darà la seguente esecuzione:

```
>./a.out < in
bimbo occorre 1 volta
ba occorre 1 volta
bi occorre 1 volta
bo occorre 1 volta
bu occorre 1 volta
bimba occorre 1 volta
bi occorre 2 volte
ba occorre 2 volte
ba occorre 3 volte
ba occorre 4 volte
bumba occorre 1 volta
bidi occorre 1 volta
bodi occorre 1 volta
bim occorre 1 volta
bum occorre 1 volta
bam occorre 1 volta

WORDTREE:

ba: 4
bam: 1
bi: 2
bidi: 1
bim: 1
bimba: 1
bimbo: 1
bo: 1
bodi: 1
bu: 1
bum: 1
bumba: 1
```

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

4 **esercizio4.cc**

```
using namespace std;

#include <iostream>
#include <cstdio>
#include <cstring>

#include "wordtree.h"

int main() {
  char prefix[MAXLENGTH]= "";
  wordtree t;
  int occ;
  char parola [MAXLENGTH];
  init(t);
  while (cin >>  parola) {
    aggiungi_parola(parola,t);
    occ = occorrenze_parola(parola,t);
    cout << parola << " occorre " << occ
         << ((occ==1) ? " volta\n" : " volte\n");
  }
  cout << "\nWORDTREE: \n\n";
  stampa_wordtree(t,prefix);
  return 0;
}

// Definire qui sotto init, aggiungi_parola e occorrenze_parola

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


int index(char c) {
  return (c-'a');
}

void aggiungi_parola(char * parola, wordtree & t) {
  int res;
  char c = *parola;
  if (t==NULL) {
    t = new node;
    for (int i=0;i<DIM;i++)
      t->subtree[i]=NULL;
    t->occorrenze=0;
  }
  if (c =='\0')
    t->occorrenze++;
  else {
    parola++;
    aggiungi_parola(parola, t->subtree[index(c)]);
  }
}
```

```
int occorrenze_parola(char * parola, wordtree & t) {
  int res;
  char c = *parola;
  if (t==NULL)
    res=0;
  else if (c ==’\0’)
    res = t-> occorrenze;
  else {
    parola++;
    res= occorrenze_parola(parola,t->subtree[index(c)]);
  }
  return res;
}
```