Secondo Appello di Programmazione I

11 Febbraio 2014
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 i nomi di due file di testo:

- legga le parole contenute nel primo file,
- ne estragga le parole che possano potenzialmente essere interpretate come indirizzi email,
- stampi tali parole nel secondo file.

Si assuma per semplicità che il file di ingresso contenga righe di lunghezza non superiore a 80 caratteri. È importante ricordare che tutte le parole nel testo sono sempre separate dal testo circostante tramite separatori.

Per essere riconosciuta come indirizzo email (semplificato rispetto ad un caso reale), una parola deve avere le seguenti caratteristiche:

- deve contenere uno ed uno solo simbolo "`@`" che divida la parola in due parole, nella forma: `<nomeutente>@<dominio>`
- entrambe le parole `<dominio>` e `<nomeutente>` devono:
  - avere lunghezza ≥ 1
  - essere formate esclusivamente da
    - lettere minuscole `a-z`,
    - lettere maiuscole `A-Z`,
    - cifre `0-9`,
    - i simboli punto `.` e underscore `_`;
  - non avere il simbolo punto `.` come prima o ultima lettera.

Ad esempio "`.nome@gmail.com`", "`mio-nome@gmail.com`", "`nome@gmail.com.`" non sono riconosciute come indirizzi email validi. Se viene dato in input il file di testo presente nella directory, il file in output conterrà:

```
a@b
abc@1
cristoforo.colombo@aol.com
GiuseppeVerdi@gmail.com
Marco_polo@yahoo.cn
galileogalilei1564@unitn.it
Leonardo.da.Vinci@gmail.com
f1bonacci@dmath.unitn.it
```

NOTA 1: **NON** è possibile utilizzare le funzioni della libreria `cstring`

NOTA 2: Va implementato il controllo dei parametri in ingresso, in particolare il file in input deve essere esistente ed accessibile in lettura.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

const int MAX_LENGTH = 80;

// Ritorna l'indice della prima occorrenza di simbolo
// in stringa, o l'indice del primo '\0' se non trovato.
// E' possibile utilizzare questa funzione anche per ricercare
// l'indice del carattere terminatore.
//
int index_of(const char *stringa, char simbolo) {
  int i;
  for (i=0; (stringa[i]!=simbolo) && (stringa[i]!='\0'); i++)
     ;
  return i;
}

// Ritorna true se il carattere e' ammesso nella
// parola, false altrimenti.
//
bool is_allowed(char simbolo) {
    return ((simbolo >= 'a' && simbolo <= 'z')
            || (simbolo >= 'A' && simbolo <= 'Z')
            || (simbolo >= '0' && simbolo <= '9')
            || (simbolo == '.')
            || (simbolo == '_'));
}

// Controlla se la parola compresa tra gli indici start/end
// e' valida. Ritorna true se la parola e' valida, false altrimenti
//
bool is_word(const char * word, int start, int end) {
    bool res = true;
    if (end < start) {
        // Gli indici che delimitano la parola devono essere validi
        // La parola deve essere lunga almeno un caratteri
        res = false;
    }
    else if ((word[start] == '.') || (word[end] == '.')) {
        // Una parola non puo' iniziare o finire con un punto
        res = false;
    }
    else {
    // Controllo tutti i caratteri (compreso l'ultimo)
      for (int i = start; i <= end; i++) {
          if (!is_allowed(word[i])) {
             // Trovato carattere non permesso
              res = false;
      }
```

```
      }
    }

    return res;
}

// Controlla se la stringa e' un indirizzo email valido
///
bool is_mail(char stringa[]) {
    bool res;
    int at = index_of(stringa, '@');
    int terminatore = index_of(stringa, '\0');
    if (at==terminatore) // la stringa non contiene un "@"
        res = false;
    else {
        if ((is_word(stringa, 0, at-1)) &&
            (is_word(stringa, at+1, terminatore-1))) {
            res = true;
        }
    }
    return res;
}

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

    // Controllo argomenti passati in ingresso
    if (argc != 3) {
        cerr << "Sintassi: ./a.out <myin> <myout>." << endl;
        exit(EXIT_FAILURE);
    }

    // Tentativo di apertura file di input
    myin.open(argv[1], ios::in);
    if (myin.fail()) {
        cerr << "Il file " << argv[1] << " non esiste o non e' accessibile.\n";
        exit(EXIT_FAILURE);
    }
    myout.open(argv[2], ios::out);

    char token[MAX_LENGTH];
    myin >> token;
    while (!myin.eof()) {
        if (is_mail(token)) {
            // L'indirizzo e' valido
            myout << token << endl;
        }
        myin >> token;
    }
    // Chiude i file
    myin.close();
    myout.close();

    return 0;
```

```
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti i nomi di due file di testo:

- legga le parole contenute nel primo file,
- ne estragga le parole che possano potenzialmente essere interpretate come indirizzi http,
- stampi tali parole nel secondo file.

Si assuma per semplicità che il file di ingresso contenga righe di lunghezza non superiore a 80 caratteri. È importante ricordare che tutte le parole nel testo sono sempre separate dal testo circostante tramite separatori.

Per essere riconosciuta come indirizzo http (semplificato rispetto ad un caso reale), una parola deve avere le seguenti caratteristiche:

- deve iniziare con la sequenza `http://` che divida la parola in due parti, nella forma: `http://<path>`
- in cui `<path>` deve:
  - avere lunghezza ≥ 1
  - essere formato esclusivamente da
    - lettere minuscole `a-z`,
    - lettere maiuscole `A-Z`,
    - cifre `0-9`,
    - i simboli dash `-`, slash `/` e punto `.`;
  - non avere il simbolo punto `.` come prima o ultima lettera.

Ad esempio:
`disi.unitn.it`
`http://disi_unitn.it`
`http://.disi.unitn.it`
`http://disi.unitn.it.`

non sono riconosciuti come indirizzi http validi. Se viene dato in input il file di testo presente nella directory, il file in output conterrà:

```
http://a
http://disi.unitn.it
http://google.com/
http://www.unitn.it/scienze/4464/laurea-informatica
http://events.unitn.it/porteaperte2014
http://www3.unitn.it/EN/scienze/6790/bachelor-degree-in-computer-science
http://unitn.it/en/ingegneria/home.html
http://it.wikipedia.org/wiki/Http
```

NOTA 1: **NON** è possibile utilizzare le funzioni della libreria `cstring`

NOTA 2: Va implementato il controllo dei parametri in ingresso, in particolare il file in input deve essere esistente ed accessibile in lettura.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

const int MAX_LENGTH = 80;

// Ritorna il numero di caratteri iniziali che
//le due stringhe hanno in comune. La funzione e' case sensitive
int starts_with(const char *stringa, const char *sub) {
  int i;
  for (i=0; stringa[i]==sub[i]; i++)
     ;
  return i;
}

// Ritorna true se il carattere e' ammesso nella
// parola, false altrimenti.
//
bool is_allowed(char simbolo) {
    return ((simbolo >= 'a' && simbolo <= 'z')
            || (simbolo >= 'A' && simbolo <= 'Z')
            || (simbolo >= '0' && simbolo <= '9')
            || (simbolo == '.')
            || (simbolo == '-')
            || (simbolo == '/'));
}

// Controlla se la parola compresa tra gli indici start/end
// e' valida. Ritorna true se la parola e' valida, false altrimenti
//
bool is_word(const char * word, int start, int end) {
    bool res = true;
    if (end < start) {
        // Gli indici che delimitano la parola devono essere validi
        // La parola deve essere lunga almeno un caratteri
        res = false;
    }
    else if ((word[start] == '.') || (word[end] == '.')) {
        // Una parola non puo' iniziare o finire con un punto
        res = false;
    }
    else {
    // Controllo tutti i caratteri (compreso l'ultimo)
      for (int i = start; i <= end; i++) {
          if (!is_allowed(word[i])) {
             // Trovato carattere non permesso
              res = false;
      }
      }
    }
```

```
    return res;
}

// Controlla se la stringa e' un indirizzo http valido
///
bool is_http(char stringa[]) {
    bool res = false;
    int match = starts_with(stringa, "http://");
    int terminatore;

    //cerco l'indice del terminatore
    for (terminatore=0; stringa[terminatore]!='\0'; terminatore++)
      ;

    if (match<7) // la stringa non inizia con "http://"
        res = false;
    else {
        if (is_word(stringa, match, terminatore-1)) {
            res = true;
        }
    }
    return res;
}

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

    // Controllo argomenti passati in ingresso
    if (argc != 3) {
        cerr << "Sintassi: ./a.out <myin> <myout>." << endl;
        exit(EXIT_FAILURE);
    }

    // Tentativo di apertura file di input
    myin.open(argv[1], ios::in);
    if (myin.fail()) {
        cerr << "Il file " << argv[1] << " non esiste o non e' accessibile.\n";
        exit(EXIT_FAILURE);
    }
    myout.open(argv[2], ios::out);

    char token[MAX_LENGTH];
    myin >> token;
    while (!myin.eof()) {
        if (is_http(token)) {
            // L'indirizzo e' valido
            myout << token << endl;
        }
        myin >> token;
    }
    // Chiude i file
    myin.close();
    myout.close();
```

```
    return 0;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti i nomi di due file di testo:

- legga le parole contenute nel primo file,
- ne estragga le parole che possano potenzialmente essere interpretate come indirizzi https,
- stampi tali parole nel secondo file.

Si assuma per semplicità che il file di ingresso contenga righe di lunghezza non superiore a 80 caratteri. È importante ricordare che tutte le parole nel testo sono sempre separate dal testo circostante tramite separatori.

Per essere riconosciuta come indirizzo https (semplificato rispetto ad un caso reale), una parola deve avere le seguenti caratteristiche:

- deve iniziare con la sequenza `https://` che divida la parola in due parti, nella forma: `https://<path>`
- in cui `<path>` deve:
  - avere lunghezza ≥ 1
  - essere formato esclusivamente da
    - lettere minuscole `a-z`,
    - lettere maiuscole `A-Z`,
    - cifre `0-9`,
    - i simboli dash `-`, slash `/` e punto `.`;
  - non avere il simbolo punto `.` come prima o ultima lettera.

Ad esempio:
`disi.unitn.it`
`https://disi_unitn.it`
`https://.disi.unitn.it`
`https://disi.unitn.it.`

non sono riconosciuti come indirizzi https validi. Se viene dato in input il file di testo presente nella directory, il file in output conterrà:

```
https://a
https://disi.unitn.it
https://google.com/
https://www.unitn.it/scienze/4464/laurea-informatica
https://events.unitn.it/porteaperte2014
https://www3.unitn.it/EN/scienze/6790/bachelor-degree-in-computer-science
https://unitn.it/en/ingegneria/home.html
https://it.wikipedia.org/wiki/Https
```

NOTA 1: **NON** è possibile utilizzare le funzioni della libreria `cstring`

NOTA 2: Va implementato il controllo dei parametri in ingresso, in particolare il file in input deve essere esistente ed accessibile in lettura.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

const int MAX_LENGTH = 80;

// Ritorna il numero di caratteri iniziali che
//le due stringhe hanno in comune. La funzione e' case sensitive
int starts_with(const char *stringa, const char *sub) {
  int i;
  for (i=0; stringa[i]==sub[i]; i++)
     ;
  return i;
}

// Ritorna true se il carattere e' ammesso nella
// parola, false altrimenti.
//
bool is_allowed(char simbolo) {
    return ((simbolo >= 'a' && simbolo <= 'z')
            || (simbolo >= 'A' && simbolo <= 'Z')
            || (simbolo >= '0' && simbolo <= '9')
            || (simbolo == '.')
            || (simbolo == '-')
            || (simbolo == '/'));
}

// Controlla se la parola compresa tra gli indici start/end
// e' valida. Ritorna true se la parola e' valida, false altrimenti
//
bool is_word(const char * word, int start, int end) {
    bool res = true;
    if (end < start) {
        // Gli indici che delimitano la parola devono essere validi
        // La parola deve essere lunga almeno un caratteri
        res = false;
    }
    else if ((word[start] == '.') || (word[end] == '.')) {
        // Una parola non puo' iniziare o finire con un punto
        res = false;
    }
    else {
    // Controllo tutti i caratteri (compreso l'ultimo)
      for (int i = start; i <= end; i++) {
          if (!is_allowed(word[i])) {
             // Trovato carattere non permesso
              res = false;
      }
      }
    }
```

```
    return res;
}

// Controlla se la stringa e' un indirizzo https valido
///
bool is_https(char stringa[]) {
    bool res = false;
    int match = starts_with(stringa, "https://");
    int terminatore;

    //cerco l'indice del terminatore
    for (terminatore=0; stringa[terminatore]!='\0'; terminatore++)
      ;

    if (match<8) // la stringa non inizia con "https://"
        res = false;
    else {
        if (is_word(stringa, match, terminatore-1)) {
            res = true;
        }
    }
    return res;
}

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

    // Controllo argomenti passati in ingresso
    if (argc != 3) {
        cerr << "Sintassi: ./a.out <myin> <myout>." << endl;
        exit(EXIT_FAILURE);
    }

    // Tentativo di apertura file di input
    myin.open(argv[1], ios::in);
    if (myin.fail()) {
        cerr << "Il file " << argv[1] << " non esiste o non e' accessibile.\n";
        exit(EXIT_FAILURE);
    }
    myout.open(argv[2], ios::out);

    char token[MAX_LENGTH];
    myin >> token;
    while (!myin.eof()) {
        if (is_https(token)) {
            // L'indirizzo e' valido
            myout << token << endl;
        }
        myin >> token;
    }
    // Chiude i file
    myin.close();
    myout.close();
```

```
    return 0;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti i nomi di due file di testo:

- legga le parole contenute nel primo file,
- ne estragga le parole che possano potenzialmente essere interpretate come indirizzi ftp,
- stampi tali parole nel secondo file.

Si assuma per semplicità che il file di ingresso contenga righe di lunghezza non superiore a 80 caratteri. È importante ricordare che tutte le parole nel testo sono sempre separate dal testo circostante tramite separatori.

Per essere riconosciuta come indirizzo ftp (semplificato rispetto ad un caso reale), una parola deve avere le seguenti caratteristiche:

- deve iniziare con la sequenza `ftp://` che divida la parola in due parti, nella forma: `ftp://<path>`
- in cui `<path>` deve:
  - avere lunghezza ≥ 1
  - essere formato esclusivamente da
    - lettere minuscole `a-z`,
    - lettere maiuscole `A-Z`,
    - cifre `0-9`,
    - i simboli dash `-`, slash `/` e punto `.`;
  - non avere il simbolo punto `.` come prima o ultima lettera.

Ad esempio:

```
disi.unitn.it
ftp://disi_unitn.it
ftp://.disi.unitn.it
ftp://disi.unitn.it.
```

non sono riconosciuti come indirizzi ftp validi. Se viene dato in input il file di testo presente nella directory, il file in output conterrà:

```
ftp://a
ftp://disi.unitn.it
ftp://google.com/
ftp://public.unitn.it/scienze/4464/laurea-informatica.pdf
ftp://events.unitn.it/porteaperte2014.html
ftp://unitn.it/EN/scienze/6790/computer-science.pdf
ftp://unitn.it/en/ingegneria/programma.txt
ftp://it.wikipedia.org/wiki/Https.pdf
```

NOTA 1: **NON** è possibile utilizzare le funzioni della libreria `cstring`

NOTA 2: Va implementato il controllo dei parametri in ingresso, in particolare il file in input deve essere esistente ed accessibile in lettura.

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

1 **esercizio1.cc**

```
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

const int MAX_LENGTH = 80;

// Ritorna il numero di caratteri iniziali che
//le due stringhe hanno in comune. La funzione e' case sensitive
int starts_with(const char *stringa, const char *sub) {
  int i;
  for (i=0; stringa[i]==sub[i]; i++)
     ;
  return i;
}

// Ritorna true se il carattere e' ammesso nella
// parola, false altrimenti.
//
bool is_allowed(char simbolo) {
    return ((simbolo >= 'a' && simbolo <= 'z')
            || (simbolo >= 'A' && simbolo <= 'Z')
            || (simbolo >= '0' && simbolo <= '9')
            || (simbolo == '.')
            || (simbolo == '-')
            || (simbolo == '/'));
}

// Controlla se la parola compresa tra gli indici start/end
// e' valida. Ritorna true se la parola e' valida, false altrimenti
//
bool is_word(const char * word, int start, int end) {
    bool res = true;
    if (end < start) {
        // Gli indici che delimitano la parola devono essere validi
        // La parola deve essere lunga almeno un caratteri
        res = false;
    }
    else if ((word[start] == '.') || (word[end] == '.')) {
        // Una parola non puo' iniziare o finire con un punto
        res = false;
    }
    else {
    // Controllo tutti i caratteri (compreso l'ultimo)
      for (int i = start; i <= end; i++) {
          if (!is_allowed(word[i])) {
             // Trovato carattere non permesso
              res = false;
      }
      }
    }
```

```
    return res;
}

// Controlla se la stringa e' un indirizzo ftp valido
///
bool is_ftp(char stringa[]) {
    bool res = false;
    int match = starts_with(stringa, "ftp://");
    int terminatore;

    //cerco l'indice del terminatore
    for (terminatore=0; stringa[terminatore]!='\0'; terminatore++)
      ;

    if (match<6) // la stringa non inizia con "ftp://"
        res = false;
    else {
        if (is_word(stringa, match, terminatore-1)) {
            res = true;
        }
    }
    return res;
}

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

    // Controllo argomenti passati in ingresso
    if (argc != 3) {
        cerr << "Sintassi: ./a.out <myin> <myout>." << endl;
        exit(EXIT_FAILURE);
    }

    // Tentativo di apertura file di input
    myin.open(argv[1], ios::in);
    if (myin.fail()) {
        cerr << "Il file " << argv[1] << " non esiste o non e' accessibile.\n";
        exit(EXIT_FAILURE);
    }
    myout.open(argv[2], ios::out);

    char token[MAX_LENGTH];
    myin >> token;
    while (!myin.eof()) {
        if (is_ftp(token)) {
            // L'indirizzo e' valido
            myout << token << endl;
        }
        myin >> token;
    }
    // Chiude i file
    myin.close();
    myout.close();
```

```
    return 0;
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `estrai` che, data una stringa lunga al massimo 80 caratteri senza, estrae tutte le lettere maiuscole contenute e restituisce un'altra stringa contenente le sole lettere maiuscole estratte.

Per esempio, data la stringa $"RobertoAlessandroMarioEnrico"$, la stringa che dovrà essere estratta e restituita sarà $"RAME"$.

NOTA 1: **La funzione `estrai` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Si può fare uso di eventuali funzioni ricorsive ausiliarie.**

NOTA 2: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

const int DIM = 81;

// Dichiarare qui sotto la funzione estrai
char *estrai(char stringa[]);
void estrai_ric(char stringa[], int i, char app[], int i_app);

int main () {
   char stringa[DIM];
   char  * estratta;
   char risposta='n';

   do {
     cout << "Inserisci la stringa da controllare: ";
     cin >> stringa;

     estratta = estrai(stringa);

     cout << "La stringa estratta e': ";
     cout << estratta;
     cout << endl;
     cout << "Vuoi inserire un'altra stringa? [s/n] ";
     cin >> risposta;
   } while (risposta != 'n');

   return 0;
}


// Definire qui sotto la funzione estrai

char *estrai(char stringa[]) {
   char * app = new char[DIM];

   estrai_ric(stringa,0,app,0);

   return (app);
}

void estrai_ric(char stringa[], int i, char app[], int i_app) {
   if (stringa[i]=='\0') {
     app[i_app]='\0';
   } else {
      if ((stringa[i]>='A')&&(stringa[i]<='Z')) {
     app[i_app]=stringa[i];
             estrai_ric(stringa,i+1,app,i_app+1);
          }
      else {
             estrai_ric(stringa,i+1,app,i_app);
          }
```

```
    }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `estrai` che, data una stringa lunga al massimo 80 caratteri senza, estrae tutte le lettere minuscole contenute e restituisce un'altra stringa contenente le sole lettere minuscole estratte.

Per esempio, data la stringa $''Programmazione1''$, la stringa che dovrà essere estratta e restituita sarà $''rogrammazione''$.

NOTA 1: **La funzione `estrai` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Si può fare uso di eventuali funzioni ricorsive ausiliarie.**

NOTA 2: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

const int DIM = 81;

// Dichiarare qui sotto la funzione estrai
char *estrai(char stringa[]);
void estrai_ric(char stringa[], int i, char app[], int i_app);

int main () {
   char stringa[DIM];
   char  * estratta;
   char risposta='n';

   do {
     cout << "Inserisci la stringa da controllare: ";
     cin >> stringa;

     estratta = estrai(stringa);

     cout << "La stringa estratta e': ";
     cout << estratta;
     cout << endl;
     cout << "Vuoi inserire un'altra stringa? [s/n] ";
     cin >> risposta;
   } while (risposta != 'n');

   return 0;
}


// Definire qui sotto la funzione estrai

char *estrai(char stringa[]) {
   char * app = new char[DIM];

   estrai_ric(stringa,0,app,0);

   return (app);
}

void estrai_ric(char stringa[], int i, char app[], int i_app) {
   if (stringa[i]=='\0') {
     app[i_app]='\0';
   } else {
      if ((stringa[i]>='a')&&(stringa[i]<='z')) {
     app[i_app]=stringa[i];
             estrai_ric(stringa,i+1,app,i_app+1);
          }
      else {
             estrai_ric(stringa,i+1,app,i_app);
          }
```

```
    }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `estrai` che, data una stringa lunga al massimo 80 caratteri senza, estrae tutte le cifre contenute e restituisce un'altra stringa contenente le sole cifre estratte.

Per esempio, data la stringa $''Programmazione1''$, la stringa che dovrà essere estratta e restituita sarà $''1''$.

NOTA 1: **La funzione `estrai` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Si può fare uso di eventuali funzioni ricorsive ausiliarie.**

NOTA 2: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

const int DIM = 81;

// Dichiarare qui sotto la funzione estrai
char *estrai(char stringa[]);
void estrai_ric(char stringa[], int i, char app[], int i_app);

int main () {
   char stringa[DIM];
   char  * estratta;
   char risposta='n';

   do {
     cout << "Inserisci la stringa da controllare: ";
     cin >> stringa;

     estratta = estrai(stringa);

     cout << "La stringa estratta e': ";
     cout << estratta;
     cout << endl;
     cout << "Vuoi inserire un'altra stringa? [s/n] ";
     cin >> risposta;
   } while (risposta != 'n');

   return 0;
}


// Definire qui sotto la funzione estrai

char *estrai(char stringa[]) {
   char * app = new char[DIM];

   estrai_ric(stringa,0,app,0);

   return (app);
}

void estrai_ric(char stringa[], int i, char app[], int i_app) {
   if (stringa[i]=='\0') {
     app[i_app]='\0';
   } else {
      if ((stringa[i]>='0')&&(stringa[i]<='9')) {
     app[i_app]=stringa[i];
            estrai_ric(stringa,i+1,app,i_app+1);
         }
      else {
            estrai_ric(stringa,i+1,app,i_app);
         }
```

```
    }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `estrai` che, data una stringa lunga al massimo 80 caratteri senza, estrae tutte le cifre contenute e restituisce un'altra stringa contenente le sole cifre estratte.

Per esempio, data la stringa $''Anno2014''$, la stringa che dovrà essere estratta e restituita sarà $''2014''$.

NOTA 1: **La funzione `estrai` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Si può fare uso di eventuali funzioni ricorsive ausiliarie.**

NOTA 2: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **esercizio2.cc**

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

const int DIM = 81;

// Dichiarare qui sotto la funzione estrai
char *estrai(char stringa[]);
void estrai_ric(char stringa[], int i, char app[], int i_app);

int main () {
   char stringa[DIM];
   char  * estratta;
   char risposta='n';

   do {
     cout << "Inserisci la stringa da controllare: ";
     cin >> stringa;

     estratta = estrai(stringa);

     cout << "La stringa estratta e': ";
     cout << estratta;
     cout << endl;
     cout << "Vuoi inserire un'altra stringa? [s/n] ";
     cin >> risposta;
   } while (risposta != 'n');

   return 0;
}


// Definire qui sotto la funzione estrai

char *estrai(char stringa[]) {
   char * app = new char[DIM];

   estrai_ric(stringa,0,app,0);

   return (app);
}

void estrai_ric(char stringa[], int i, char app[], int i_app) {
   if (stringa[i]=='\0') {
     app[i_app]='\0';
   } else {
      if ((stringa[i]>='0')&&(stringa[i]<='9')) {
     app[i_app]=stringa[i];
             estrai_ric(stringa,i+1,app,i_app+1);
          }
      else {
             estrai_ric(stringa,i+1,app,i_app);
          }
```

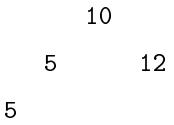
```
    }
}
```

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

- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **crescente** e se l'elemento è già presente deve essere inserito a **sinistra**. Esempio: l'inserimento dei seguenti valori:

  ```
  10  5  12  5
  ```

  deve produrre il seguente albero:

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

  ```
  5  5  10  12
  ```

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

3 **albero_main.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

int main() {
  char res;
  Albero albero;
  int val;
  inizializza(albero);
  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 (inserisci(albero, val) == FALSO) {
        cout << "Valore gia` presente!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (cerca(albero, val) == VERO) {
        cout << "Valore presente: " << val << endl;
      } else {
        cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (vuoto(albero) == VERO) {
        cout << "Albero vuoto!" << endl;
      } else {
        stampa(albero);
      }
      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 Nodo {
    int val;
    Nodo *sx;
    Nodo *dx;
};

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, int val);
boolean cerca(const Albero &t, int val);
void stampa(const Albero &t);

#endif
```

3 **soluzione_A31.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

void inizializza(Albero &a) {
  a = NULL;
}

boolean vuoto(const Albero &a) {
  return (a == NULL) ? VERO : FALSO;
}

boolean inserisci(Albero &a, int val) {
  // Caso base
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a == NULL) {
       return FALSO;
    }
    a->val = val;
    a->sx = a->dx = NULL;
    return VERO;
  }
  // Caso ricorsivo. Controllo se scendere a sinistra o a destra
  if (val <= a->val) {
    // Scendo a sinistra
    return inserisci(a->sx, val);
```

```
  } else if (val > a->val) {
    // Scendo a destra
    return inserisci(a->dx, val);
  }
}

boolean cerca(const Albero &a, int val) {
  if (vuoto(a) == VERO) {
    return FALSO;
  } else if (val == a->val) {
    // Trovato
    return VERO;
  } else if (val < a->val) {
    // Scendo a sinistra
    return cerca(a->sx, val);
  } else {
    // Scendo a destra
    return cerca(a->dx, val);
  }
}

void stampa(const Albero &a) {
  if (vuoto(a) == FALSO) {
    // Prima stampo gli elementi MINORI di a->val (cioe' quelli a sx)
    stampa(a->sx);
    // Poi stampo a->val
    cout << a-> val << ' ';
    // Poi stampo gli elementi MAGGIORI (cioe' quelli a dx)
    stampa(a->dx);
  }
}
```

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

- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **decrescente** e se l'elemento è già presente deve essere inserito a **sinistra**. Esempio: l'inserimento dei seguenti valori:

  ```
  9  12  5  12
  ```

  deve produrre il seguente albero:

  ```
          9
      12      5
  12
  ```

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

  ```
  5  5  9  12
  ```

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

3 **albero_main.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

int main() {
  char res;
  Albero albero;
  int val;
  inizializza(albero);
  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 (inserisci(albero, val) == FALSO) {
        cout << "Valore gia` presente!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (cerca(albero, val) == VERO) {
        cout << "Valore presente: " << val << endl;
      } else {
        cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (vuoto(albero) == VERO) {
        cout << "Albero vuoto!" << endl;
      } else {
        stampa(albero);
      }
      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 Nodo {
    int val;
    Nodo *sx;
    Nodo *dx;
};

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, int val);
boolean cerca(const Albero &t, int val);
void stampa(const Albero &t);

#endif
```

3 **soluzione_A32.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

void inizializza(Albero &a) {
  a = NULL;
}

boolean vuoto(const Albero &a) {
  return (a == NULL) ? VERO : FALSO;
}

boolean inserisci(Albero &a, int val) {
  // Caso base
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a == NULL) {
       return FALSO;
    }
    a->val = val;
    a->sx = a->dx = NULL;
    return VERO;
  }
  // Caso ricorsivo. Controllo se scendere a sinistra o a destra
  if (val < a->val) {
    // Scendo a destra
    return inserisci(a->dx, val);
```

```
  } else if (val >= a->val) {
    // Scendo a sinistra
    return inserisci(a->sx, val);
  }
}

boolean cerca(const Albero &a, int val) {
  if (vuoto(a) == VERO) {
    return FALSO;
  } else if (val == a->val) {
    // Trovato
    return VERO;
  } else if (val < a->val) {
    // Scendo a destra
    return cerca(a->dx, val);
  } else {
    // Scendo a sinistra
    return cerca(a->sx, val);
  }
}

void stampa(const Albero &a) {
  if (vuoto(a) == FALSO) {
    // Prima stampo gli elementi MINORI di a->val (cioe' quelli a dx)
    stampa(a->dx);
    // Poi stampo a->val
    cout << a-> val << ' ';
    // Poi stampo gli elementi MAGGIORI (cioe' quelli a sx)
    stampa(a->sx);
  }
}
```

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

- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **crescente** e se l'elemento è già presente deve essere inserito a **destra**. Esempio: l'inserimento dei seguenti valori:

  ```
  7  8  6  8
  ```

  deve produrre il seguente albero:

  ```
        7
    6       8
              8
  ```

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

  ```
  6  7  8  8
  ```

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

3 **albero_main.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

int main() {
  char res;
  Albero albero;
  int val;
  inizializza(albero);
  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 (inserisci(albero, val) == FALSO) {
        cout << "Valore gia` presente!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (cerca(albero, val) == VERO) {
        cout << "Valore presente: " << val << endl;
      } else {
        cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (vuoto(albero) == VERO) {
        cout << "Albero vuoto!" << endl;
      } else {
        stampa(albero);
      }
      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 Nodo {
    int val;
    Nodo *sx;
    Nodo *dx;
};

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, int val);
boolean cerca(const Albero &t, int val);
void stampa(const Albero &t);

#endif
```

3 **soluzione_A33.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

void inizializza(Albero &a) {
  a = NULL;
}

boolean vuoto(const Albero &a) {
  return (a == NULL) ? VERO : FALSO;
}

boolean inserisci(Albero &a, int val) {
  // Caso base
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a == NULL) {
       return FALSO;
    }
    a->val = val;
    a->sx = a->dx = NULL;
    return VERO;
  }
  // Caso ricorsivo. Controllo se scendere a sinistra o a destra
  if (val < a->val) {
    // Scendo a sinistra
    return inserisci(a->sx, val);
```

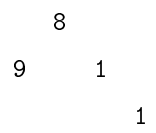
```
  } else if (val >= a->val) {
    // Scendo a destra
    return inserisci(a->dx, val);
  }
}

boolean cerca(const Albero &a, int val) {
  if (vuoto(a) == VERO) {
    return FALSO;
  } else if (val == a->val) {
    // Trovato
    return VERO;
  } else if (val < a->val) {
    // Scendo a sinistra
    return cerca(a->sx, val);
  } else {
    // Scendo a destra
    return cerca(a->dx, val);
  }
}

void stampa(const Albero &a) {
  if (vuoto(a) == FALSO) {
    // Prima stampo gli elementi MINORI di a->val (cioe' quelli a sx)
    stampa(a->sx);
    // Poi stampo a->val
    cout << a-> val << ' ';
    // Poi stampo gli elementi MAGGIORI (cioe' quelli a dx)
    stampa(a->dx);
  }
}
```

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

- `inizializza` inizializzi l'albero;
- `vuoto` controlli se l'albero è vuoto, restituendo `VERO` in caso affermativo e `FALSO` in caso contrario;
- `inserisci` inserisca l'elemento passato come parametro nell'albero, restituendo `VERO` se l'operazione è andata a buon fine, e `FALSO` altrimenti. L'albero deve essere ordinato in maniera **decrescente** e se l'elemento è già presente deve essere inserito a **destra**. Esempio: l'inserimento dei seguenti valori:

  ```
  8  9  1  1
  ```

  deve produrre il seguente albero:

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

  ```
  1  1  8  9
  ```

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

3 **albero_main.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

int main() {
  char res;
  Albero albero;
  int val;
  inizializza(albero);
  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 (inserisci(albero, val) == FALSO) {
        cout << "Valore gia` presente!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (cerca(albero, val) == VERO) {
        cout << "Valore presente: " << val << endl;
      } else {
        cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (vuoto(albero) == VERO) {
        cout << "Albero vuoto!" << endl;
      } else {
        stampa(albero);
      }
      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 Nodo {
    int val;
    Nodo *sx;
    Nodo *dx;
};

typedef Nodo * Albero;

enum boolean { VERO, FALSO };

void inizializza(Albero &t);
boolean vuoto(const Albero &t);
boolean inserisci(Albero &t, int val);
boolean cerca(const Albero &t, int val);
void stampa(const Albero &t);

#endif
```

3 **soluzione_A34.cc**

```
#include <iostream>
#include "albero.h"

using namespace std;

void inizializza(Albero &a) {
  a = NULL;
}

boolean vuoto(const Albero &a) {
  return (a == NULL) ? VERO : FALSO;
}

boolean inserisci(Albero &a, int val) {
  // Caso base
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a == NULL) {
       return FALSO;
    }
    a->val = val;
    a->sx = a->dx = NULL;
    return VERO;
  }
  // Caso ricorsivo. Controllo se scendere a sinistra o a destra
  if (val <= a->val) {
    // Scendo a destra
    return inserisci(a->dx, val);
```

```
  } else if (val > a->val) {
    // Scendo a sinistra
    return inserisci(a->sx, val);
  }
}

boolean cerca(const Albero &a, int val) {
  if (vuoto(a) == VERO) {
    return FALSO;
  } else if (val == a->val) {
    // Trovato
    return VERO;
  } else if (val < a->val) {
    // Scendo a destra
    return cerca(a->dx, val);
  } else {
    // Scendo a sinistra
    return cerca(a->sx, val);
  }
}

void stampa(const Albero &a) {
  if (vuoto(a) == FALSO) {
    // Prima stampo gli elementi MINORI di a->val (cioe' quelli a dx)
    stampa(a->dx);
    // Poi stampo a->val
    cout << a-> val << ' ';
    // Poi stampo gli elementi MAGGIORI (cioe' quelli a sx)
    stampa(a->sx);
  }
}
```

4 Scrivere all'interno del file `esercizio4.cc` la definizione della procedura `ordina` che, presi come parametri in ingresso un'array di interi `v` ed un intero `n`, corrispondente al numero di elementi ivi contenuti, ordini il contenuto dell'array stesso in modo `crescente`, implementando l'algoritmo di ordinamento `QUICKSORT`.

NOTA 1: La funzione `ordina` deve essere ricorsiva: una funzione è ricorsiva se invoca se stessa oppure se invoca altre funzioni ricorsive.

NOTA 2: Non è ammesso l'utilizzo di cicli iterativi nell'implementazione di `ordina`.

NOTA 3: È ammessa la dichiarazione e la definizione di funzioni ausiliarie.

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