Primo Appello di Programmazione I

13 Febbraio 2018
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, preso come unico argomento della main il nome di un file di testo, legga le parole contenute nel file e ne estragga gli indirizzi email, stampandoli in output.

È importante notare che gli indirizzi email nel testo sono sempre separati dal testo circostante tramite spazi.

Un indirizzo email per essere valido deve avere inoltre le seguenti caratteristiche (semplificate rispetto ad un caso reale):

- l'indirizzo email contiene uno ed uno solo simbolo "at" (`@`) che divide il nomeutente dal dominio in questo modo: `nomeutente@dominio`
- il nomeutente ed il dominio sono parole per le quali vale:
  - la lunghezza è maggiore o uguale a due caratteri
  - può contenere lettere nell'intervallo (`a-z`) (lettere minuscole)
  - può contenere lettere nell'intervallo (`A-Z`) (lettere maiuscole)
  - può contenere numeri nell'intervallo (`0-9`) (numeri)
  - può contenere il simbolo underscore (_)
  - può contenere il simbolo punto (`.`), ma non come prima o ultima lettera. Ad esempio `.nome@gmail.com` non e' considerato un indirizzo email valido e nemmeno `nome@gmail.com.`

L'output del programma dovrà essere uguale a:

```
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 librerie `string.h`, `cstring` o affini.

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

NOTA 3: La stringa contenente la parola letta dal file può essere un array di char statico. Si consiglia di utilizzare una lunghezza di 81 caratteri.

NOTA 4: A titolo di esempio, è possibile utilizzare il file di input che si trova nella directory dell'esercizio.

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

1 **soluzione_A11.cc**

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

using namespace std;

const int MAX_LENGTH = 81;

int index_of(const char *stringa, char simbolo);
bool is_allowed(char simbolo);
bool valid_word(const char * word, int start, int end);
bool is_mail(char stringa[]);

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

  // Controllo argomenti passati in ingresso
  if (argc != 2) {
    cerr << "Sintassi: ./a.out <in>." << endl;
    exit(EXIT_FAILURE);
  }

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

  char token[MAX_LENGTH];
  in >> token;
  while (!in.eof()) {
    if (is_mail(token)) {
      // L'indirizzo e' valido
      cout << token << endl;
    }
    in >> token;
  }

  // Chiude il file
  in.close();

  return 0;
}

// Ritorna l'indice della prima occorrenza di simbolo
// in stringa, o -1 se non trovata.
// Il primo carattere ha indice pari a 0 (zero)
// E' possibile utilizzare questa funzione anche per ricercare
// l'indice del carattere terminatore.
int index_of(const char *stringa, char simbolo) {
  int i = 0;
```

```
  while (stringa[i] != simbolo && stringa[i] != '\0' && i < MAX_LENGTH) {
    i++;
  }
  return stringa[i] == simbolo ? i : (simbolo == '\0' ? MAX_LENGTH - 1 : -1);
}

// 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 valid_word(const char * word, int start, int end) {
  bool ris = true;
  if ((start < 0) || (end <= start)) {
    // Gli indici che delimitano la parola devono essere validi
    // La parola deve essere lunga almeno due caratteri
    ris = false;
  }

  if ((word[start] == '.') || (word[end] == '.')) {
    // Una parola non puo' iniziare o finire con un punto
    ris = false;
  }
  // Controllo tutti i caratteri (compreso l'ultimo)
  for (int i = start; i <= end; i++) {
    if (!is_allowed(word[i])) {
      // Trovato carattere non permesso
      ris = false;
    }
  }
  return ris;
}

// Controlla se la stringa e' un indirizzo email valido
bool is_mail(char stringa[]) {
  bool ris = false;
  int lun = index_of(stringa, '\0');
  int at = index_of(stringa, '@');
  // Se le stringa contiene un carattere at (@) al suo interno
  if (at > -1) {
    if ((valid_word(stringa, 0, at - 1))
        && (valid_word(stringa, at + 1, terminatore - 1))) {
      ris = true;
    }
  }
  return ris;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, preso come unico argomento della main il nome di un file di testo, legga le parole contenute nel file e ne estragga gli indirizzi di siti web, stampandoli in output.

È importante notare che gli indirizzi di siti web nel testo sono sempre separati dal testo circostante tramite spazi.

Un indirizzo web per essere valido deve avere inoltre le seguenti caratteristiche (semplificate rispetto ad un caso reale):

- l'indirizzo web comincia con la sequenza di caratteri "`http://`" che precede l'indirizzo vero e proprio, il quale ha la forma: `dominio/percorso`
- dominio e percorso sono a loro volta separati tra loro dal carattere "`/`".
- il dominio e il percorso sono parole per le quali vale:
  - possono contenere lettere nell'intervallo (`a-z`) (lettere minuscole)
  - possono contenere lettere nell'intervallo (`A-Z`) (lettere maiuscole)
  - possono contenere numeri nell'intervallo (`0-9`) (numeri)
  - possono contenere il simbolo underscore (_)
- inoltre:
  - il dominio ha lunghezza maggiore o uguale a tre caratteri, mentre il percorso può anche avere lunghezza nulla
  - il dominio può contenere il simbolo punto (`.`), ma non il simbolo "`/`"; viceversa, il percorso può contenere il simbolo "`/`", ma non il simbolo punto.

L'output del programma dovrà essere uguale a:

```
http://www.unitn.it/
http://aeroplane/engine/bolt
http://a.b/c
http://strano_ma_valido/0123456789
http://www.google.com/a
http://abcde/first/second/third
```

NOTA 1: **NON** è possibile utilizzare le funzioni della librerie `string.h`, `cstring` o affini.

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

NOTA 3: La stringa contenente la parola letta dal file può essere un array di char statico. Si consiglia di utilizzare una lunghezza di 81 caratteri.

NOTA 4: A titolo di esempio, è possibile utilizzare il file di input che si trova nella directory dell'esercizio.

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

1 **soluzione_A12.cc**

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

using namespace std;

const int MAX_LENGTH = 81;
const int PREAMBLE_LENGTH = 7;
const char PREAMBLE[PREAMBLE_LENGTH + 1] = "http://";
const char DOMAIN_SEPARATOR = '.';
const char PATH_SEPARATOR = '/';

int index_of(const char stringa[], char simbolo);
int index_of(const char stringa[], int offset, char simbolo);
bool startsWith(const char ricerca[], const char in[]);
bool is_allowed(char simbolo);
bool is_valid_domain(const char stringa[], int offset, int end);
bool is_valid_path(const char stringa[], int offset, int end);
bool is_valid_url(const char stringa[]);

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

  // Controllo argomenti passati in ingresso
  if (argc != 2) {
    cerr << "Sintassi: ./a.out <in>." << endl;
    exit(EXIT_FAILURE);
  }

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

  char token[MAX_LENGTH];
  in >> token;
  while (!in.eof()) {
    if (is_valid_url(token)) {
      // L'indirizzo e' valido
      cout << token << endl;
    }
    in >> token;
  }

  // Chiude il file
  in.close();

  return 0;
}
```

```
int index_of(const char stringa[], char simbolo) {
  return index_of(stringa, 0, simbolo);
}

int index_of(const char stringa[], int offset, char simbolo) {
  // Ritorna l'indice della prima occorrenza di simbolo
  // in stringa, o -1 se non trovata.
  // Il primo carattere ha indice pari a 0 (zero)
  // E' possibile utilizzare questa funzione anche per ricercare
  // l'indice del carattere terminatore
  int i = offset;
  // Attenzione: non controlla se "offset" e'
  // minore della lunghezza della stringa
  while (stringa[i] != simbolo && stringa[i] != '\0' && i < MAX_LENGTH) {
    i++;
  }
  return stringa[i] == simbolo ? i : (simbolo == '\0' ? MAX_LENGTH - 1 : -1);
}

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

bool startsWith(const char ricerca[], const char in[]) {
  // Ritorna vero se e solo se "in" comincia
  // con "ricerca"
  bool ris = false;
  // Calcola la lunghezza delle due stringhe
  int lun_ricerca = index_of(ricerca, '\0'),
      lun_in = index_of(in, '\0');
  if(lun_ricerca <= lun_in) {
    ris = true;
    for(int i = 0; i < lun_ricerca && ris; i++) {
      if(ricerca[i] != in[i]) {
        ris = false;
      }
    }
  }
  return ris;
}

bool is_valid_domain(const char stringa[], int offset, int end) {
  bool ris = false;
  // Lunghezza minima: 3 caratteri
  if(end - offset >= 2) {
    ris = true;
    for(int i = offset; i < end && ris; i++) {
      // Non e' nessuno dei caratteri ammessi
      if(!is_allowed(stringa[i]) && stringa[i] != DOMAIN_SEPARATOR) {
```

```
        ris = false;
      }
    }
  }
  return ris;
}

bool is_valid_path(const char stringa[], int offset, int end) {
  bool ris = false;
  // Puo' anche essere vuoto
  if(end - offset >= 0) {
    ris = true;
    for(int i = offset; i < end && ris; i++) {
      // Non e' nessuno dei caratteri ammessi
      if(!is_allowed(stringa[i]) && stringa[i] != PATH_SEPARATOR) {
        ris = false;
      }
    }
  }
  return ris;
}

bool is_valid_url(const char stringa[]) {
  bool ris = false;
  // Calcola la lunghezza della stringa
  int len = index_of(stringa, '\0');
  // Deve cominciare con "http://"
  if(startsWith(PREAMBLE, stringa)) {
    int firstSeparator = index_of(stringa, PREAMBLE_LENGTH, PATH_SEPARATOR);
    // Il nome del dominio deve essere valido
    if(is_valid_domain(stringa, PREAMBLE_LENGTH + 1, firstSeparator)) {
      // Il percorso alla risorsa deve essere valido
      if(is_valid_path(stringa, firstSeparator + 1, len)) {
        ris = true;
      }
    }
  }
  return ris;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, preso come unico argomento della main il nome di un file di testo, legga le parole contenute nel file e ne estragga dei "tag" twitter validi, stampandoli in output.

È importante notare che questi tag nel testo sono sempre separati dal testo circostante tramite spazi.

Un tag per essere valido deve avere inoltre le seguenti caratteristiche (semplificate rispetto ad un caso reale):

- ogni tag comincia con un simbolo "hash" (`#`) o con un simbolo "at" (`@`) nel primo caso si parla di "hashtag", nel secondo di "mention"
- ogni hashtag è lungo almeno 5 caratteri (escluso il primo carattere)
- ogni mention è lunga almeno 7 caratteri (escluso il primo catattere)
- entrambi i tipi di tag:
  - possono contenere lettere nell'intervallo (`a-z`) (lettere minuscole)
  - possono contenere lettere nell'intervallo (`A-Z`) (lettere maiuscole)
  - possono contenere numeri nell'intervallo (`0-9`) (numeri)
  - possono contenere il simbolo underscore (_)
- ma, escluso il primo carattere, solo gli hashtag possono contenere il simbolo "at" (`@`), e solo le mention possono contenere il simbolo "hash" (`#`).

L'output del programma dovrà essere uguale a:

```
#unitn
@123456_
@twitterUser
#thisIs@valid
@strange#butvalid
#hashtag
```

NOTA 1: **NON** è possibile utilizzare le funzioni della librerie `string.h`, `cstring` o affini.

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

NOTA 3: La stringa contenente la parola letta dal file può essere un array di char statico. Si consiglia di utilizzare una lunghezza di 81 caratteri.

NOTA 4: A titolo di esempio, è possibile utilizzare il file di input che si trova nella directory dell'esercizio.

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

1 **soluzione_A13.cc**

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

using namespace std;

const int MAX_LENGTH = 81;
const char AT_SYMBOL = '@';
const char HASH_SYMBOL = '#';

int index_of(const char stringa[], char simbolo);
int index_of(const char stringa[], int offset, char simbolo);
bool is_allowed(char simbolo);
bool is_valid_mention(const char stringa[], int length);
bool is_valid_hashtag(const char stringa[], int length);
bool is_valid_tag(const char stringa[]);

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

  // Controllo argomenti passati in ingresso
  if (argc != 2) {
    cerr << "Sintassi: ./a.out <in>." << endl;
    exit(EXIT_FAILURE);
  }

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

  char token[MAX_LENGTH];
  in >> token;
  while (!in.eof()) {
    if (is_valid_tag(token)) {
      // Il tag e' valido
      cout << token << endl;
    }
    in >> token;
  }

  // Chiude il file
  in.close();

  return 0;
}

int index_of(const char stringa[], char simbolo) {
  // Ritorna l'indice della prima occorrenza di simbolo
  // in stringa, o -1 se non trovata.
```

```
  // Il primo carattere ha indice pari a 0 (zero)
  // E' possibile utilizzare questa funzione anche per ricercare
  // l'indice del carattere terminatore
  int i = 0;
  while (stringa[i] != simbolo && stringa[i] != '\0' && i < MAX_LENGTH) {
    i++;
  }
  return stringa[i] == simbolo ? i : (simbolo == '\0' ? MAX_LENGTH - 1 : -1);
}

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

bool is_valid_hashtag(const char stringa[], int length) {
  bool ris = false;
  // Lunghezza minima: 5 caratteri
  if(length >= 5) {
    ris = true;
    for(int i = 0; i < length && ris; i++) {
      // Non e' nessuno dei caratteri ammessi
      if(!is_allowed(stringa[i]) && stringa[i] != AT_SYMBOL) {
        ris = false;
      }
    }
  }
  return ris;
}

bool is_valid_mention(const char stringa[], int length) {
  bool ris = false;
  // Deve essere lungo almeno 7 caratteri
  if(length >= 7) {
    ris = true;
    for(int i = 0; i < length && ris; i++) {
      // Non e' nessuno dei caratteri ammessi
      if(!is_allowed(stringa[i]) && stringa[i] != HASH_SYMBOL) {
        ris = false;
      }
    }
  }
  return ris;
}

bool is_valid_tag(const char stringa[]) {
  bool ris = false;
  // Calcola la lunghezza della stringa
  int len = index_of(stringa, '\0');
  // Deve cominciare con "#" oppure "@"
```

```
  if(HASH_SYMBOL == stringa[0]) {
    // L'hashtag deve essere valido
    if(is_valid_hashtag(stringa + 1, len - 1)) {
      ris = true;
    }
  } else if(AT_SYMBOL == stringa[0]) {
    // La mention deve essere valida
    if(is_valid_mention(stringa + 1, len - 1)) {
      ris = true;
    }
  }
  return ris;
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, preso come unico argomento della main il nome di un file di testo, legga le parole contenute nel file e ne estragga gli indirizzi di siti web, stampandoli in output.

È importante notare che gli indirizzi di siti web nel testo sono sempre separati dal testo circostante tramite spazi.

Un indirizzo web per essere valido deve avere inoltre le seguenti caratteristiche (semplificate rispetto ad un caso reale):

- l'indirizzo web comincia con la sequenza di caratteri "`https://`" che precede l'indirizzo vero e proprio, il quale ha la forma: `dominio/percorso`
- dominio e percorso sono a loro volta separati tra loro dal carattere "`/`".
- il dominio e il percorso sono parole per le quali vale:
  - possono contenere lettere nell'intervallo (`a-z`) (lettere minuscole)
  - possono contenere lettere nell'intervallo (`A-Z`) (lettere maiuscole)
  - possono contenere numeri nell'intervallo (`0-9`) (numeri)
  - possono contenere il simbolo underscore (_)
- inoltre:
  - il dominio ha lunghezza maggiore o uguale a due caratteri, mentre il percorso può anche avere lunghezza nulla
  - il dominio può contenere il simbolo punto (`.`), ma non il simbolo "`/`"; viceversa, il percorso può contenere il simbolo "`/`", ma non il simbolo punto.

L'output del programma dovrà essere uguale a:

```
https://www.google.com/a
https://a.b/c
https://secure_connection.com/
https://strano_ma_valido/0123456789
https://example.com/
https://www3.a.b.c/w/e
```

NOTA 1: **NON** è possibile utilizzare le funzioni della librerie `string.h`, `cstring` o affini.

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

NOTA 3: La stringa contenente la parola letta dal file può essere un array di char statico. Si consiglia di utilizzare una lunghezza di 81 caratteri.

NOTA 4: A titolo di esempio, è possibile utilizzare il file di input che si trova nella directory dell'esercizio.

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

1 **soluzione_A14.cc**

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

using namespace std;

const int MAX_LENGTH = 81;
const int PREAMBLE_LENGTH = 8;
const char PREAMBLE[PREAMBLE_LENGTH + 1] = "https://";
const char DOMAIN_SEPARATOR = '.';
const char PATH_SEPARATOR = '/';

int index_of(const char stringa[], char simbolo);
int index_of(const char stringa[], int offset, char simbolo);
bool startsWith(const char ricerca[], const char in[]);
bool is_allowed(char simbolo);
bool is_valid_domain(const char stringa[], int offset, int end);
bool is_valid_path(const char stringa[], int offset, int end);
bool is_valid_url(const char stringa[]);

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

  // Controllo argomenti passati in ingresso
  if (argc != 2) {
    cerr << "Sintassi: ./a.out <in>." << endl;
    exit(EXIT_FAILURE);
  }

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

  char token[MAX_LENGTH];
  in >> token;
  while (!in.eof()) {
    if (is_valid_url(token)) {
      // L'indirizzo e' valido
      cout << token << endl;
    }
    in >> token;
  }

  // Chiude il file
  in.close();

  return 0;
}
```

```
int index_of(const char stringa[], char simbolo) {
  return index_of(stringa, 0, simbolo);
}

int index_of(const char stringa[], int offset, char simbolo) {
  // Ritorna l'indice della prima occorrenza di simbolo
  // in stringa, o -1 se non trovata.
  // Il primo carattere ha indice pari a 0 (zero)
  // E' possibile utilizzare questa funzione anche per ricercare
  // l'indice del carattere terminatore
  int i = offset;
  // Attenzione: non controlla se "offset" e'
  // minore della lunghezza della stringa
  while (stringa[i] != simbolo && stringa[i] != '\0' && i < MAX_LENGTH) {
    i++;
  }
  return stringa[i] == simbolo ? i : (simbolo == '\0' ? MAX_LENGTH - 1 : -1);
}

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

bool startsWith(const char ricerca[], const char in[]) {
  // Ritorna vero se e solo se "in" comincia
  // con "ricerca"
  bool ris = false;
  // Calcola la lunghezza delle due stringhe
  int lun_ricerca = index_of(ricerca, '\0'),
      lun_in = index_of(in, '\0');
  if(lun_ricerca <= lun_in) {
    ris = true;
    for(int i = 0; i < lun_ricerca && ris; i++) {
      if(ricerca[i] != in[i]) {
        ris = false;
      }
    }
  }
  return ris;
}

bool is_valid_domain(const char stringa[], int offset, int end) {
  bool ris = false;
  // Lunghezza minima: 2 caratteri
  if(end - offset >= 1) {
    ris = true;
    for(int i = offset; i < end && ris; i++) {
      // Non e' nessuno dei caratteri ammessi
      if(!is_allowed(stringa[i]) && stringa[i] != DOMAIN_SEPARATOR) {
```

```
        ris = false;
      }
    }
  }
  return ris;
}

bool is_valid_path(const char stringa[], int offset, int end) {
  bool ris = false;
  // Puo' anche essere vuoto
  if(end - offset >= 0) {
    ris = true;
    for(int i = offset; i < end && ris; i++) {
      // Non e' nessuno dei caratteri ammessi
      if(!is_allowed(stringa[i]) && stringa[i] != PATH_SEPARATOR) {
        ris = false;
      }
    }
  }
  return ris;
}

bool is_valid_url(const char stringa[]) {
  bool ris = false;
  // Calcola la lunghezza della stringa
  int len = index_of(stringa, '\0');
  // Deve cominciare con "http://"
  if(startsWith(PREAMBLE, stringa)) {
    int firstSeparator = index_of(stringa, PREAMBLE_LENGTH, PATH_SEPARATOR);
    // Il nome del dominio deve essere valido
    if(is_valid_domain(stringa, PREAMBLE_LENGTH + 1, firstSeparator)) {
      // Il percorso alla risorsa deve essere valido
      if(is_valid_path(stringa, firstSeparator + 1, len)) {
        ris = true;
      }
    }
  }
  return ris;
}
```

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

Per esempio, data la stringa **"RedGreenBlue"**, la stringa che dovrà essere estratta e restituita sarà **"RGB"**.

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 e/o funzioni wrapper.**

NOTA 2: la funzione definita non deve in alcun modo alterare i parametri passati in ingresso.

NOTA 3: 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 **codice_A21.cc**

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

const int DIM = 81;

// Dichiarare qui sotto la funzione estrai

int main () {
   char stringa[DIM], *estratta, risposta;

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

     estratta = estrai(stringa);

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

   return 0;
}

// Definire qui sotto la funzione estrai
```

2 **soluzione_A21.cc**

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

const int DIM = 80;

char* estrai(const char[]);
void estrai_ric(const char[], int, char[], int);

int main () {
   char stringa[DIM + 1], *estratta, risposta;

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

     estratta = estrai(stringa);

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

   return 0;
}
```

```
char* estrai(const char input[]) {
  // Allocazione dinamica
  char* output = new char[DIM + 1];
  estrai_ric(input, 0, output, 0);
  return output;
}

void estrai_ric(const char input[], int indice_input, char output[], int indice_output) {
  char c = input[indice_input];
  if(c == '\0' || indice_input == DIM) {
    // Caso base
    output[indice_output] = '\0';
  } else {
    // Estrazione caratteri filtrati
    if(c >= 'A' && c <= 'Z') {
      output[indice_output] = c;
      indice_output++;
    }
    // Chiamata ricorsiva
    estrai_ric(input, indice_input + 1, output, indice_output);
  }
}
```

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

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 e/o funzioni wrapper.**

NOTA 2: la funzione definita non deve in alcun modo alterare i parametri passati in ingresso.

NOTA 3: 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 **codice_A22.cc**

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

const int DIM = 81;

// Dichiarare qui sotto la funzione estrai

int main () {
   char stringa[DIM], *estratta, risposta;

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

     estratta = estrai(stringa);

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

   return 0;
}

// Definire qui sotto la funzione estrai
```

2 **soluzione_A22.cc**

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

const int DIM = 80;

char* estrai(const char[]);
void estrai_ric(const char[], int, char[], int);

int main () {
   char stringa[DIM + 1], *estratta, risposta;

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

     estratta = estrai(stringa);

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

   return 0;
}
```

```
char* estrai(const char input[]) {
  // Allocazione dinamica
  char* output = new char[DIM + 1];
  estrai_ric(input, 0, output, 0);
  return output;
}

void estrai_ric(const char input[], int indice_input, char output[], int indice_output) {
  char c = input[indice_input];
  if(c == '\0' || indice_input == DIM) {
    // Caso base
    output[indice_output] = '\0';
  } else {
    // Estrazione caratteri filtrati
    if(c >= 'a' && c <= 'z') {
      output[indice_output] = c;
      indice_output++;
    }
    // Chiamata ricorsiva
    estrai_ric(input, indice_input + 1, output, indice_output);
  }
}
```

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

Per esempio, data la stringa **"11Marzo2014"**, la stringa che dovrà essere estratta e restituita sarà **"112014"**.

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 e/o funzioni wrapper.**

NOTA 2: la funzione definita non deve in alcun modo alterare i parametri passati in ingresso.

NOTA 3: 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 **codice_A23.cc**

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

const int DIM = 81;

// Dichiarare qui sotto la funzione estrai

int main () {
   char stringa[DIM], *estratta, risposta;

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

     estratta = estrai(stringa);

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

   return 0;
}

// Definire qui sotto la funzione estrai
```

2 **soluzione_A23.cc**

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

const int DIM = 80;

char* estrai(const char[]);
void estrai_ric(const char[], int, char[], int);

int main () {
   char stringa[DIM + 1], *estratta, risposta;

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

     estratta = estrai(stringa);

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

   return 0;
}
```

```
char* estrai(const char input[]) {
  // Allocazione dinamica
  char* output = new char[DIM + 1];
  estrai_ric(input, 0, output, 0);
  return output;
}

void estrai_ric(const char input[], int indice_input, char output[], int indice_output) {
  char c = input[indice_input];
  if(c == '\0' || indice_input == DIM) {
    // Caso base
    output[indice_output] = '\0';
  } else {
    // Estrazione caratteri filtrati
    if(c >= '0' && c <= '9') {
      output[indice_output] = c;
      indice_output++;
    }
    // Chiamata ricorsiva
    estrai_ric(input, indice_input + 1, output, indice_output);
  }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `estrai` che, data una stringa lunga al massimo 80 caratteri, estrae tutti caratteri alfabetici, maiuscoli o minuscoli, e restituisce un'altra stringa contenente i soli caratteri estratti; i caratteri non alfabetici vanno comunque scartati.

Per esempio, data la stringa **"11Marzo2014"**, la stringa che dovrà essere estratta e restituita sarà **"Marzo"**.

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 e/o funzioni wrapper.**

NOTA 2: la funzione definita non deve in alcun modo alterare i parametri passati in ingresso.

NOTA 3: 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 **codice_A24.cc**

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

const int DIM = 81;

// Dichiarare qui sotto la funzione estrai

int main () {
   char stringa[DIM], *estratta, risposta;

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

     estratta = estrai(stringa);

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

   return 0;
}

// Definire qui sotto la funzione estrai
```

2 **soluzione_A24.cc**

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

const int DIM = 80;

char* estrai(const char[]);
void estrai_ric(const char[], int, char[], int);

int main () {
   char stringa[DIM + 1], *estratta, risposta;

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

     estratta = estrai(stringa);

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

   return 0;
}
```

```
char* estrai(const char input[]) {
  // Allocazione dinamica
  char* output = new char[DIM + 1];
  estrai_ric(input, 0, output, 0);
  return output;
}

void estrai_ric(const char input[], int indice_input, char output[], int indice_output) {
  char c = input[indice_input];
  if(c == '\0' || indice_input == DIM) {
    // Caso base
    output[indice_output] = '\0';
  } else {
    // Estrazione caratteri filtrati
    if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
      output[indice_output] = c;
      indice_output++;
    }
    // Chiamata ricorsiva
    estrai_ric(input, indice_input + 1, output, indice_output);
  }
}
```

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:

  ```
          10
      5       12
  5
  ```

- `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 **albero_.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
  boolean res = FALSO;
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a != NULL) {
      a->val = val;
      a->sx = a->dx = NULL;
      res = VERO;
    }
  } else if (val <= a->val) {
    // Caso ricorsivo: scendo a sinistra
    res = inserisci(a->sx, val);
  } else {
    // Caso ricorsivo: scendo a destra
```

```
    res = inserisci(a->dx, val);
  }
  return res;
}

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

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 **albero_.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
  boolean res = FALSO;
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a != NULL) {
      a->val = val;
      a->sx = a->dx = NULL;
      res = VERO;
    }
  } else if (val < a->val) {
    // Caso ricorsivo: scendo a destra
    res = inserisci(a->dx, val);
  } else {
    // Caso ricorsivo: scendo a sinistra
```

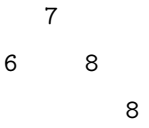
```
    res = inserisci(a->sx, val);
  }
  return res;
}

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

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:

- `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 **albero_.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
  boolean res = FALSO;
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a != NULL) {
      a->val = val;
      a->sx = a->dx = NULL;
      res = VERO;
    }
  } else if (val < a->val) {
    // Caso ricorsivo: scendo a sinistra
    res = inserisci(a->sx, val);
  } else {
    // Caso ricorsivo: scendo a destra
```

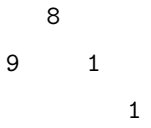
```
    res = inserisci(a->dx, val);
  }
  return res;
}

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

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 **albero_.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
  boolean res = FALSO;
  if (vuoto(a) == VERO) {
    a = new (nothrow) Nodo;
    if (a != NULL) {
      a->val = val;
      a->sx = a->dx = NULL;
      res = VERO;
    }
  } else if (val <= a->val) {
    // Caso ricorsivo: scendo a destra
    res = inserisci(a->dx, val);
  } else {
    // Caso ricorsivo: scendo a sinistra
```

```
    res = inserisci(a->sx, val);
  }
  return res;
}

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

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.

4 **soluzione_A41.cc**

```
#include <iostream>
#include <iomanip>

using namespace std;

const int DIM = 16;

void printarray(int v[], int min, int max);
void swap(int & a, int & b);
void sposta(int v[], int p, int u, int & piv);
void ordina1(int v[], int primo, int ultimo);
void ordina(int v[], int n);

int main() {
  // Esempio
  int myarray[DIM] = {41,3,9,1,5,17,6,20,37,2,8,23,10,0,11,19};

  printarray(myarray, 0, DIM - 1);
  cout << endl;
  ordina(myarray, DIM);
  printarray(myarray, 0, DIM - 1);
}

void printarray(int v[], int min, int max) {
  int i;
  cout << "[";
  for (i= min; i <= max;i++) {
    cout << setw(2) << v[i] << " ";
  }
  cout << "]\n";
}

void swap(int & a, int & b) {
  int c = a;
  a = b;
  b = c;
}

void sposta(int v[], int p, int u, int & piv) {
  if (p >= u) {
    swap(v[p], v[piv]);
    piv=p;
  } else if (v[p] <= v[piv]) {
    sposta(v, p + 1, u, piv);
  } else if (v[u] >= v[piv]) {
    sposta(v, p, u - 1, piv);
  } else { // v[p]>v[piv]>v[u]
    swap(v[p], v[u]);
    sposta(v, p + 1, u - 1, piv);
  }
}
```

```
void ordina1(int v[], int primo, int ultimo) {
  if (primo < ultimo) {
    int piv = ultimo;
    sposta(v, primo, ultimo, piv);
    ordina1(v, primo, piv-1);
    ordina1(v, piv + 1, ultimo);
  }
}

void ordina(int v[],int n) {
  ordina1(v, 0, n - 1);
}
```