Primo Appello di Programmazione I

09 gennaio 2019
Prof. Roberto Sebastiani

Codice:

Nome	Cognome	Matricola

La directory 'esame' contiene 4 sotto-directory: 'uno', 'due', 'tre' e 'quattro'. Le soluzioni vanno scritte negli spazi e nei modi indicati esercizio per esercizio. NOTA: il codice dato non può essere modificato

Modalità di questo appello

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

- Non è consentito l'uso di alcun libro di testo o fotocopia. In caso lo studente necessitasse di carta (?), gli/le verranno forniti fogli di carta bianca su richiesta, che dovranno essere riconsegnati a fine prova. È consentito l'uso di una penna. Non è consentito l'uso di alcuno strumento calcolatore.
- È vietato lo scambio di qualsiasi informazione, orale o scritta. È vietato guardare nel terminale del vicino.
- È vietato l'uso di telefoni cellulari o di qualsiasi strumento elettronico.
- È vietato allontanarsi dall'aula durante la prova, anche se si ha già consegnato. (Ogni necessità fisiologica va espletata PRIMA dell'inizio della prova.)
- È vietato qualunque accesso, in lettura o scrittura, a file esterni alla directory di lavoro assegnata a ciascun studente. Le uniche operazioni consentite sono l'apertura, l'editing, la copia, la rimozione e la compilazione di file all'interno della propria directory di lavoro.
- Sono ovviamente vietati l'uso di email, ftp, ssh, telnet ed ogni strumento che consenta di accedere a file esterni alla directory di lavoro. Le operazioni di copia, rimozione e spostamento di file devono essere circoscritte alla directory di lavoro.
- Ogni altra attività non espressamente citata qui sopra o autorizzata dal docente è vietata.

Ogni violazione delle regole di cui sopra comporterà automaticamente l'annullamento della prova e il divieto di accesso ad un certo numero di appelli successivi, a seconda della gravità e della recidività della violazione.

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

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

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

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

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

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

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

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

Figura 1: `testo`

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

Figura 2: `testocorretto`

NOTA 1: Per semplicità si assuma che il testo contenuto nel primo file contenga almeno una parola e che inizi con un carattere alfabetico, non contenga "..." e che ".", "?" e "!" siano sempre preceduti da una parola e seguiti da almeno una spaziatura. Si assuma inoltre che ogni parola contenuta nel testo del primo file abbia al massimo lunghezza 30 caratteri.

NOTA 2: Come si vede dall'esempio mostrato, non è necessario preservare in output lo stesso numero delle righe presenti in input.

NOTA 3: È ammesso l'uso della funzione `strlen` della libreria `<cstring>`.

NOTA 4: il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). Per realizzare la conversione da caratteri minuscoli in maiuscoli, è vietato l'uso di tabelle o di 26 `if` o `switch-case`, uno per ogni carattere.

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

1 **esercizio1.cc**

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

bool is_lower(char c);

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

  fstream my_in, my_out;
  char tmp[31];

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

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

  my_in >> tmp;
  int l = strlen(tmp);
  if(l > 0 && is_lower(tmp[0])) {
    tmp[0] = tmp[0] + ('A' - 'a');
  }

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

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

bool is_lower(char c) {
  return (c >= 'a' && c <= 'z');
}
```

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

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

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

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

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

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

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

Figura 3: `testo`

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

Figura 4: `testocorretto`

NOTA 1: Per semplicità si assuma che il testo contenuto nel primo file contenga almeno una parola e che inizi con un carattere alfabetico, non contenga "..." e che ",", ";" e ":" siano sempre preceduti da una parola e seguiti da almeno una spaziatura. Si assuma inoltre che ogni parola contenuta nel testo del primo file abbia al massimo lunghezza 30 caratteri.

NOTA 2: Come si vede dall'esempio mostrato, non è necessario preservare in output lo stesso numero delle righe presenti in input.

NOTA 3: È ammesso l'uso della funzione `strlen` della libreria `<cstring>`.

NOTA 4: il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). Per realizzare la conversione da caratteri minuscoli in maiuscoli, è vietato l'uso di tabelle o di 26 `if` o `switch-case`, uno per ogni carattere.

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

1 **esercizio1.cc**

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

bool is_upper(char c);

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

  fstream my_in, my_out;
  char tmp[31];

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

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

  my_in >> tmp;
  int l = strlen(tmp);
  if(l > 0 && is_upper(tmp[0])) {
    tmp[0] = tmp[0] + ('A' - 'a');
  }

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

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

bool is_upper(char c) {
  return (c >= 'A' && c <= 'Z');
}
```

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

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

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

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

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

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

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

Figura 5: `testo`

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

Figura 6: `testocorretto`

NOTA 1: Per semplicità si assuma che il testo contenuto nel primo file contenga almeno una parola e che inizi con un carattere alfabetico, non contenga "..." e che ".", "?" e "!" siano sempre preceduti da una parola e seguiti da almeno una spaziatura. Si assuma inoltre che ogni parola contenuta nel testo del primo file abbia al massimo lunghezza 30 caratteri.

NOTA 2: Come si vede dall'esempio mostrato, non è necessario preservare in output lo stesso numero delle righe presenti in input.

NOTA 3: È ammesso l'uso della funzione `strlen` della libreria `<cstring>`.

NOTA 4: il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). Per realizzare la conversione da caratteri minuscoli in maiuscoli, è vietato l'uso di tabelle o di 26 `if` o `switch-case`, uno per ogni carattere.

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

1 **esercizio1.cc**

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

bool is_lower(char c);
bool is_upper(char c);

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

  fstream my_in, my_out;
  char tmp[31];

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

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

  my_in >> tmp;
  int l = strlen(tmp);
  if(l > 0 && is_lower(tmp[0])) {
    tmp[0] = tmp[0] + ('A' - 'a');
  }

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

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

bool is_upper(char c) {
  return (c >= 'A' && c <= 'Z');
}

bool is_lower(char c) {
```

```
  return (c >= 'a' && c <= 'z');
}
```

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

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

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

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

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

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

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

Figura 7: `testo`

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

Figura 8: `testocorretto`

NOTA 1: Per semplicità si assuma che il testo contenuto nel primo file contenga almeno una parola e che inizi con un carattere alfabetico, non contenga "..." e che ",", ";" e ":" siano sempre preceduti da una parola e seguiti da almeno una spaziatura. Si assuma inoltre che ogni parola contenuta nel testo del primo file abbia al massimo lunghezza 30 caratteri.

NOTA 2: Come si vede dall'esempio mostrato, non è necessario preservare in output lo stesso numero delle righe presenti in input.

NOTA 3: È ammesso l'uso della funzione `strlen` della libreria `<cstring>`.

NOTA 4: il programma deve potenzialmente funzionare con ogni possibile codifica dei caratteri secondo le regole di tali codifiche viste a lezione (quindi non solo ASCII). Per realizzare la conversione da caratteri minuscoli in maiuscoli, è vietato l'uso di tabelle o di 26 `if` o `switch-case`, uno per ogni carattere.

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

1 **esercizio1.cc**

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

bool is_upper(char c);
bool is_lower(char c);

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

  fstream my_in, my_out;
  char tmp[31];

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

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

  my_in >> tmp;
  int l = strlen(tmp);
  if(l > 0 && is_upper(tmp[0])) {
    tmp[0] = tmp[0] + ('A' - 'a');
  }

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

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

bool is_upper(char c) {
  return (c >= 'A' && c <= 'Z');
}

bool is_lower(char c) {
```

```
  return (c >= 'a' && c <= 'z');
}
```

2 Si definiscono "*primi*" quei numeri interi che non hanno altri divisori al di fuori di 1 e di sé stessi. Si conviene che 1 *non* sia un numero primo.

All'interno del programma "`esercizio2.cc`" è dato, come costante globale, l'array "`CENTO_PRIMI`" contenente l'elenco ordinato dei primi 100 numeri primi.

Completare il programma definito nel file `esercizio2.cc` implementando la funzione ricorsiva "**primo**" che riceva in input un intero n > 1, e che utilizzi l'array CENTO_PRIMI come segue:

- restituisce "**1**" se n e' un elemento dell'array;
- restituisce "**0**" se n non e' un elemento dell'array ma e' multiplo di un elemento dell'array;
- restituisce "**-1**" altrimenti.

La funzione **primo** deve essere ricorsiva: non sono ammessi nella sua implementazione nè cicli, nè invocazioni ad altre funzioni che non siano a loro volta ricorsive. È ammessa la dichiarazione e l'implementazione di eventuali funzioni ausiliarie (wrapper) qualora ritenute necessarie alla soluzione dell'esercizio.

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

2 **esercizio2.cc**

```
#include <iostream>

using namespace std;

const int MAX_PRIMI = 100;
const int CENTO_PRIMI[MAX_PRIMI] =
{  2,   3,   5,   7,  11,  13,  17,  19,  23,  29,
  31,  37,  41,  43,  47,  53,  59,  61,  67,  71,
  73,  79,  83,  89,  97, 101, 103, 107, 109, 113,
 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
 233, 239, 241, 251, 257, 263, 269, 271, 277, 281,
 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
 419, 421, 431, 433, 439, 443, 449, 457, 461, 463,
 467, 479, 487, 491, 499, 503, 509, 521, 523, 541};

int primo(int n);
int primoRic(int n, int indice);

int main() {
  int n,ris;
  do {
    cout << "Introdurre un numero intero > 1: ";
    cin >> n;
    if (n < 1)
      cerr << "Il numero e' troppo piccolo." << endl;
    else {
      ris = primo(n);
      if (ris < 0)
        cout << "Non sono riuscito a stabilire se " << n << " e' primo\n";
      else
        cout << n << ((ris > 0) ? "" : " non") << " e' primo." << endl;
    }
  } while (n > 1);
  return (0);
}

int primoRic(int n, int indice) {
  int ris;
  if (indice >= MAX_PRIMI)         // fine scansione dell'array
    ris = -1;
  else if (n == CENTO_PRIMI[indice]) // n e' un elemento dell'array
    ris = 1;
  else if ((n % CENTO_PRIMI[indice]) == 0) // n e' multiplo di un
    ris = 0;                               // elemento dell'array
  else {
    ris = primoRic(n, indice+1);
  }
  return ris;
}
```

```
int primo(int n) {
  return primoRic(n,  0);
}
```

2 Si definiscono "*primi*" quei numeri interi che non hanno altri divisori al di fuori di 1 e di sé stessi. Si conviene che 1 *non* sia un numero primo.

All'interno del programma "`esercizio2.cc`" è dato, come costante globale, l'array "`CENTO_PRIMI`" contenente l'elenco ordinato dei primi 100 numeri primi.

Completare il programma definito nel file `esercizio2.cc` implementando la funzione ricorsiva "**primo**" che riceva in input un intero n > 1, e che utilizzi l'array CENTO_PRIMI come segue:

- restituisce "**1**" se n e' un elemento dell'array;
- se n non e' un elemento dell'array ma e' multiplo di (almeno) un elemento dell'array, restituisce il più piccolo divisore trovato (es.: se introduco "**26**", otterrò "**2**", se introduco "**91**", otterrò "**7**");
- restituisce "**-1**" altrimenti.

La funzione **primo** deve essere ricorsiva: non sono ammessi nella sua implementazione nè cicli, nè invocazioni ad altre funzioni che non siano a loro volta ricorsive. È ammessa la dichiarazione e l'implementazione di eventuali funzioni ausiliarie (wrapper) qualora ritenute necessarie alla soluzione dell'esercizio.

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

2 **esercizio2.cc**

```
#include <iostream>

using namespace std;

const int MAX_PRIMI = 100;
const int CENTO_PRIMI[MAX_PRIMI] =
{  2,   3,   5,   7,  11,  13,  17,  19,  23,  29,
  31,  37,  41,  43,  47,  53,  59,  61,  67,  71,
  73,  79,  83,  89,  97, 101, 103, 107, 109, 113,
 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
 233, 239, 241, 251, 257, 263, 269, 271, 277, 281,
 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
 419, 421, 431, 433, 439, 443, 449, 457, 461, 463,
 467, 479, 487, 491, 499, 503, 509, 521, 523, 541};

int primo(int n);
int primoRic(int n, int indice);

int main() {
  int n,ris;
  do {
    cout << "Introdurre un numero intero > 1: ";
    cin >> n;
    if (n < 1)
      cerr << "Il numero e' troppo piccolo." << endl;
    else {
      ris = primo(n);
      if (ris < 0)
        cout << "Non sono riuscito a stabilire se " << n << " e' primo\n";
      else
        if(ris == 1)
          cout << n << " e' primo." << endl;
        else
          cout << n << " e' multiplo di " << ris << "." << endl;
    }
  } while (n > 1);
  return (0);
}

int primoRic(int n, int indice) {
  int ris;
  if (indice >= MAX_PRIMI)         // fine scansione dell'array
    ris = -1;
  else if (n == CENTO_PRIMI[indice]) // n e' un elemento dell'array
    ris = 1;
  else if ((n % CENTO_PRIMI[indice]) == 0) // n e' multiplo di un
    ris = CENTO_PRIMI[indice];             // elemento dell'array
  else {
    ris = primoRic(n, indice+1);
  }
```

```
  return ris;
}

int primo(int n) {
  return primoRic(n,  0);
}
```

2 Si definiscono "*primi*" quei numeri interi che non hanno altri divisori al di fuori di 1 e di sé stessi. Si conviene che 1 *non* sia un numero primo.

All'interno del programma "`esercizio2.cc`" è dato, come costante globale, l'array "`CENTO_PRIMI`" contenente l'elenco ordinato dei primi 100 numeri primi.

Completare il programma definito nel file `esercizio2.cc` implementando la funzione ricorsiva "**primo**" che riceva in input un intero n > 1, e che utilizzi l'array CENTO_PRIMI come segue:

- restituisce "**-2**" se n e' un elemento dell'array;
- se n non e' un elemento dell'array ma e' multiplo di (almeno) un elemento dell'array, restituisce l'indice del più piccolo divisore trovato (es.: se introduco "**26**", otterrò "**0**", se introduco "**91**", otterrò "**3**");
- restituisce "**-1**" altrimenti.

La funzione **primo** deve essere ricorsiva: non sono ammessi nella sua implementazione nè cicli, nè invocazioni ad altre funzioni che non siano a loro volta ricorsive. È ammessa la dichiarazione e l'implementazione di eventuali funzioni ausiliarie (wrapper) qualora ritenute necessarie alla soluzione dell'esercizio.

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

2 **esercizio2.cc**

```
#include <iostream>

using namespace std;

const int MAX_PRIMI = 100;
const int CENTO_PRIMI[MAX_PRIMI] =
{  2,   3,   5,   7,  11,  13,  17,  19,  23,  29,
  31,  37,  41,  43,  47,  53,  59,  61,  67,  71,
  73,  79,  83,  89,  97, 101, 103, 107, 109, 113,
 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
 233, 239, 241, 251, 257, 263, 269, 271, 277, 281,
 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
 419, 421, 431, 433, 439, 443, 449, 457, 461, 463,
 467, 479, 487, 491, 499, 503, 509, 521, 523, 541};

int primo(int n);
int primoRic(int n, int indice);

int main() {
  int n,ris;
  do {
    cout << "Introdurre un numero intero > 1: ";
    cin >> n;
    if (n < 1)
      cerr << "Il numero e' troppo piccolo." << endl;
    else {
      ris = primo(n);
      switch(ris) {
        case -1:
          cout << "Non sono riuscito a stabilire se " << n << " e' primo\n";
          break;
        case -2:
          cout << n << " e' primo." << endl;
          break;
        default:
          cout << n << " e' multiplo di " << CENTO_PRIMI[ris] << "." << endl;
      }
    }
  } while (n > 1);
  return (0);
}

int primoRic(int n, int indice) {
  int ris;
  if (indice >= MAX_PRIMI)         // fine scansione dell'array
    ris = -1;
  else if (n == CENTO_PRIMI[indice]) // n e' un elemento dell'array
    ris = -2;
  else if ((n % CENTO_PRIMI[indice]) == 0) // n e' multiplo di un
    ris = indice;                          // elemento dell'array
```

```
  else {
    ris = primoRic(n, indice+1);
  }
  return ris;
}

int primo(int n) {
  return primoRic(n,  0);
}
```

2 Si definiscono "*primi*" quei numeri interi che non hanno altri divisori al di fuori di 1 e di sé stessi. Si conviene che 1 *non* sia un numero primo.

All'interno del programma "`esercizio2.cc`" è dato, come costante globale, l'array "`CENTO_PRIMI`" contenente l'elenco ordinato dei primi 100 numeri primi.

Completare il programma definito nel file `esercizio2.cc` implementando la funzione ricorsiva "**primo**" che riceva in input un intero n > 1, e che utilizzi l'array CENTO_PRIMI come segue:

- restituisce "**1**" se n e' un elemento dell'array;
- se n non e' un elemento dell'array ma e' multiplo di (almeno) un elemento dell'array, restituisce il quoziente della divisione tra n e l'elemento trovato (es.: se introduco "**26**", otterrò "**13**", se introduco "**91**", otterrò "**13**");
- restituisce "**-1**" altrimenti.

La funzione **primo** deve essere ricorsiva: non sono ammessi nella sua implementazione nè cicli, nè invocazioni ad altre funzioni che non siano a loro volta ricorsive. È ammessa la dichiarazione e l'implementazione di eventuali funzioni ausiliarie (wrapper) qualora ritenute necessarie alla soluzione dell'esercizio.

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

2 **esercizio2.cc**

```
#include <iostream>

using namespace std;

const int MAX_PRIMI = 100;
const int CENTO_PRIMI[MAX_PRIMI] =
{  2,   3,   5,   7,  11,  13,  17,  19,  23,  29,
  31,  37,  41,  43,  47,  53,  59,  61,  67,  71,
  73,  79,  83,  89,  97, 101, 103, 107, 109, 113,
 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
 233, 239, 241, 251, 257, 263, 269, 271, 277, 281,
 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
 419, 421, 431, 433, 439, 443, 449, 457, 461, 463,
 467, 479, 487, 491, 499, 503, 509, 521, 523, 541};

int primo(int n);
int primoRic(int n, int indice);

int main() {
  int n,ris;
  do {
    cout << "Introdurre un numero intero > 1: ";
    cin >> n;
    if (n < 1)
      cerr << "Il numero e' troppo piccolo." << endl;
    else {
      ris = primo(n);
      if (ris < 0)
        cout << "Non sono riuscito a stabilire se " << n << " e' primo\n";
      else
        if(ris == 1)
          cout << n << " e' primo." << endl;
        else
          cout << n << " e' multiplo di " << n / ris << "." << endl;
    }
  } while (n > 1);
  return (0);
}

int primoRic(int n, int indice) {
  int ris;
  if (indice >= MAX_PRIMI)         // fine scansione dell'array
    ris = -1;
  else if (n == CENTO_PRIMI[indice]) // n e' un elemento dell'array
    ris = 1;
  else if ((n % CENTO_PRIMI[indice]) == 0) // n e' multiplo di un
    ris = n / CENTO_PRIMI[indice];         // elemento dell'array
  else {
    ris = primoRic(n, indice+1);
  }
```

```
  return ris;
}

int primo(int n) {
  return primoRic(n,  0);
}
```

3 Nel file `queue_main.cc` è definita la funzione `main` che contiene un menù per gestire una coda di `float`. Scrivere, in un nuovo file `queue.cc`, le definizioni delle funzioni dichiarate nello header file `queue.h` in modo tale che:

- `init` inizializzi la coda per contenere al più un numero massimo di elementi `maxdim` passato come parametro;
- `deinit` liberi la memoria utilizzata dalla coda;
- `enqueue` inserisca l'elemento passato come parametro nella coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `dequeue` tolga l'elemento in testa alla coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `first` legga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `print` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente andando a capo ad ogni elemento.

La coda deve essere implementata con un array allocato dinamicamente, e il numero massimo di elementi che possono essere inseriti nella coda è specificato dall'argomento `maxdim` della funzione `init`.

NOTA: Si ricorda da quanto visto a lezione che in tutte le possibili implementazioni di una coda, le operazioni enqueue, dequeue e first devono tassativamente richiedere un numero costante di passi computazionali, indipendente dal numero di elementi contenuti nella coda!

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

3 queue_main.cc

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

int main()
{
    char res;
    float num;
    queue q;
    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire il numero massimo di elementi della coda di float: ";
        cin >> maxdim;
    }

    init(q, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << "Enqueue (e)\n"
             << "Dequeue (d)\n"
             << "First (f)\n"
             << "Print (p)\n"
             << "Quit (q)\n";
        cin >> res;
        switch (res) {
        case 'e':
            cout << "Valore: ";
            cin >> num;
            if (!enqueue(q, num)) {
                cout << "Coda piena\n";
            }
            break;
        case 'd':
            if (!dequeue(q)) {
                cout << "Coda vuota\n";
            }
            break;
        case 'f':
            if (!first(q, num)) {
                cout << "Coda vuota!\n";
            } else {
                cout << "First = " << num << endl;
            }
            break;
        case 'p':
            print(q);
            break;
        case 'q':
            break;
        default:
            cout << "Valore errato!\n";
        }
```

```
    } while (res != 'q');
    deinit(q);

    return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_QUEUE_H
#define STRUCT_QUEUE_H

struct queue {
    int head, tail;
    int dim;
    float *elem;
};

void init(queue &q, int maxdim);
void deinit(queue &q);
bool enqueue(queue &q, float n);
bool dequeue(queue &q);
bool first(queue &q, float &out);
void print(const queue &q);

#endif
```

3 **queue.cc**

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

static int next(int index, const queue &q)
{
    return (index + 1) % q.dim;
}

void init(queue &q, int maxdim)
{
    q.tail = q.head = 0;
    q.dim = maxdim+1;
    q.elem = new float[q.dim];
}

void deinit(queue &q)
{
    delete[] q.elem;
}

static bool is_empty(const queue &q)
{
    return q.tail == q.head;
}
```

```
static bool is_full(const queue &q)
{
    return next(q.tail, q) == q.head;
}

bool enqueue(queue &q, float n)
{
    bool res = false;
    if (!is_full(q)) {
      q.elem[q.tail] = n;
      q.tail = next(q.tail, q);
      res = true;
    }
    return res;
}

bool dequeue(queue &q)
{
    bool res = false;
    if (!is_empty(q)) {
      q.head = next(q.head, q);
      res = true;
    }
    return res;
}

bool first(queue &q, float &out)
{
    bool res = false;
    if (!is_empty(q)) {
      out = q.elem[q.head];
      res = true;
    }
    return res;
}

void print(const queue &q)
{
    for (int i = q.head; i != q.tail; i = next(i, q)) {
        cout << q.elem[i] << endl;
    }
    cout << endl;
}
```

3 Nel file `queue_main.cc` è definita la funzione `main` che contiene un menù per gestire una coda di `int`. Scrivere, in un nuovo file `queue.cc`, le definizioni delle funzioni dichiarate nello header file `queue.h` in modo tale che:

- `init` inizializzi la coda per contenere al più un numero massimo di elementi `maxdim` passato come parametro;
- `deinit` liberi la memoria utilizzata dalla coda;
- `accoda` inserisca l'elemento passato come parametro nella coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `testa` legga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `estrai_testa` tolga l'elemento in testa alla coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `stampa` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente andando a capo ad ogni elemento.

La coda deve essere implementata con un array allocato dinamicamente, e il numero massimo di elementi che possono essere inseriti nella coda è specificato dall'argomento `maxdim` della funzione `init`.

NOTA: Si ricorda da quanto visto a lezione che in tutte le possibili implementazioni di una coda, le operazioni enqueue, dequeue e first devono tassativamente richiedere un numero costante di passi computazionali, indipendente dal numero di elementi contenuti nella coda!

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

3 **queue_main.cc**

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

int main()
{
    char res;
    int num;
    queue q;
    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire il numero massimo di elementi della coda di interi: ";
        cin >> maxdim;
    }

    init(q, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << "Accoda (e)\n"
             << "Estrai testa (d)\n"
             << "Testa (f)\n"
             << "Stampa (p)\n"
             << "Quit (q)\n";
        cin >> res;
        switch (res) {
        case 'e':
            cout << "Valore: ";
            cin >> num;
            if (!accoda(q, num)) {
                cout << "Coda piena\n";
            }
            break;
        case 'd':
            if (!estrai_testa(q)) {
                cout << "Coda vuota\n";
            }
            break;
        case 'f':
            if (!testa(q, num)) {
                cout << "Coda vuota!\n";
            } else {
                cout << "First = " << num << endl;
            }
            break;
        case 'p':
            stampa(q);
            break;
        case 'q':
            break;
        default:
            cout << "Valore errato!\n";
        }
```

```
    } while (res != 'q');
    deinit(q);

    return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_CODA_H
#define STRUCT_CODA_H

struct queue {
    int head, tail;
    int size;
    int *elem;
};

void init(queue &q, int maxdim);
void deinit(queue &q);
bool accoda(queue &q, int n);
bool testa(queue &q, int &out);
bool estrai_testa(queue &q);
void stampa(const queue &q);

#endif
```

3 **queue.cc**

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

static int next(int index, const queue &q)
{
    return (index + 1) % q.size;
}

void init(queue &q, int maxdim)
{
    q.tail = q.head = 0;
    q.size = maxdim+1;
    q.elem = new int[q.size];
}

void deinit(queue &q)
{
    delete[] q.elem;
}

static bool vuota(const queue &q)
{
    return q.tail == q.head;
}
```

```
static bool piena(const queue &q)
{
    return next(q.tail, q) == q.head;
}

bool accoda(queue &q, int n)
{
    bool ris = false;
    if (!piena(q)) {
      q.elem[q.tail] = n;
      q.tail = next(q.tail, q);
      ris = true;
    }
    return ris;
}

bool estrai_testa(queue &q)
{
    bool ris = false;
    if (!vuota(q)) {
      q.head = next(q.head, q);
      ris = true;
    }
    return ris;
}

bool testa(queue &q, int &out)
{
    bool ris = false;
    if (!vuota(q)) {
      out = q.elem[q.head];
      ris = true;
    }
    return ris;
}

void stampa(const queue &q)
{
    for (int i = q.head; i != q.tail; i = next(i, q)) {
        cout << q.elem[i] << endl;
    }
    cout << endl;
}
```

3 Nel file `queue_main.cc` è definita la funzione `main` che contiene un menù per gestire una coda di `double`. Scrivere, in un nuovo file `queue.cc`, le definizioni delle funzioni dichiarate nello header file `queue.h` in modo tale che:

- `init` inizializzi la coda per contenere al più un numero massimo di elementi `maxdim` passato come parametro;
- `deinit` liberi la memoria utilizzata dalla coda;
- `print` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente andando a capo ad ogni elemento;
- `enqueue` inserisca l'elemento passato come parametro nella coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `first` legga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `dequeue` tolga l'elemento in testa alla coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti.

La coda deve essere implementata con un array allocato dinamicamente, e il numero massimo di elementi che possono essere inseriti nella coda è specificato dall'argomento `maxdim` della funzione `init`.

NOTA: Si ricorda da quanto visto a lezione che in tutte le possibili implementazioni di una coda, le operazioni enqueue, dequeue e first devono tassativamente richiedere un numero costante di passi computazionali, indipendente dal numero di elementi contenuti nella coda!

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

3 queue_main.cc

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

int main()
{
    char res;
    double num;
    queue q;
    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire il numero massimo di elementi della coda di double: ";
        cin >> maxdim;
    }

    init(q, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << "Enqueue (e)\n"
             << "Dequeue (d)\n"
             << "First (f)\n"
             << "Print (p)\n"
             << "Quit (q)\n";
        cin >> res;
        switch (res) {
        case 'e':
            cout << "Valore: ";
            cin >> num;
            if (!enqueue(q, num)) {
                cout << "Coda piena\n";
            }
            break;
        case 'd':
            if (!dequeue(q)) {
                cout << "Coda vuota\n";
            }
            break;
        case 'f':
            if (!first(q, num)) {
                cout << "Coda vuota!\n";
            } else {
                cout << "First = " << num << endl;
            }
            break;
        case 'p':
            print(q);
            break;
        case 'q':
            break;
        default:
            cout << "Valore errato!\n";
        }
```

```
    } while (res != 'q');
    deinit(q);

    return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_QUEUE_H
#define STRUCT_QUEUE_H

struct queue {
    int head, tail;
    int size;
    double *elem;
};

void init(queue &q, int maxdim);
void deinit(queue &q);
void print(const queue &q);
bool enqueue(queue &q, double n);
bool first(queue &q, double &out);
bool dequeue(queue &q);

#endif
```

3 **queue.cc**

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

static int next(int index, const queue &q)
{
    return (index + 1) % q.size;
}

void init(queue &q, int maxdim)
{
    q.tail = q.head = 0;
    q.size = maxdim+1;
    q.elem = new double[q.size];
}

void deinit(queue &q)
{
    delete[] q.elem;
}

static bool is_empty(const queue &q)
{
    return q.tail == q.head;
}
```

```
static bool is_full(const queue &q)
{
    return next(q.tail, q) == q.head;
}

bool enqueue(queue &q, double n)
{
    bool ris = false;
    if (!is_full(q)) {
      q.elem[q.tail] = n;
      q.tail = next(q.tail, q);
      ris = true;
    }
    return ris;
}

bool dequeue(queue &q)
{
    bool ris = false;
    if (!is_empty(q)) {
      q.head = next(q.head, q);
      ris = true;
    }
    return ris;
}

bool first(queue &q, double &out)
{
    bool ris = false;
    if (!is_empty(q)) {
      out = q.elem[q.head];
      ris = true;
    }
    return ris;
}

void print(const queue &q)
{
    for (int i = q.head; i != q.tail; i = next(i, q)) {
        cout << q.elem[i] << endl;
    }
    cout << endl;
}
```

3 Nel file `queue_main.cc` è definita la funzione `main` che contiene un menù per gestire una coda di `char`. Scrivere, in un nuovo file `queue.cc`, le definizioni delle funzioni dichiarate nello header file `queue.h` in modo tale che:

- `init` inizializzi la coda per contenere al più un numero massimo di elementi `maxdim` passato come parametro;
- `deinit` liberi la memoria utilizzata dalla coda;
- `stampa` stampi a video il contenuto della coda, dall'elemento più vecchio al più recente andando a capo ad ogni elemento;
- `testa` legga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `estrai_testa` tolga l'elemento in testa alla coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `accoda` inserisca l'elemento passato come parametro nella coda, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti.

La coda deve essere implementata con un array allocato dinamicamente, e il numero massimo di elementi che possono essere inseriti nella coda è specificato dall'argomento `maxdim` della funzione `init`.

NOTA: Si ricorda da quanto visto a lezione che in tutte le possibili implementazioni di una coda, le operazioni enqueue, dequeue e first devono tassativamente richiedere un numero costante di passi computazionali, indipendente dal numero di elementi contenuti nella coda!

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

3 **queue_main.cc**

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

int main()
{
    char res;
    char num;
    queue q;
    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire il numero massimo di elementi della coda di char: ";
        cin >> maxdim;
    }

    init(q, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << "Accoda (e)\n"
             << "Estrai testa (d)\n"
             << "Testa (f)\n"
             << "Stampa (p)\n"
             << "Quit (q)\n";
        cin >> res;
        switch (res) {
        case 'e':
            cout << "Valore: ";
            cin >> num;
            if (!accoda(q, num)) {
                cout << "Coda piena\n";
            }
            break;
        case 'd':
            if (!estrai_testa(q)) {
                cout << "Coda vuota\n";
            }
            break;
        case 'f':
            if (!testa(q, num)) {
                cout << "Coda vuota!\n";
            } else {
                cout << "First = " << num << endl;
            }
            break;
        case 'p':
            stampa(q);
            break;
        case 'q':
            break;
        default:
            cout << "Valore errato!\n";
        }
```

```
    } while (res != 'q');
    deinit(q);

    return 0;
}
```

3 **queue.h**

```
#ifndef STRUCT_CODA_H
#define STRUCT_CODA_H

struct queue {
    int head, tail;
    int dim;
    char *elem;
};

void init(queue &q, int maxdim);
void deinit(queue &q);
bool accoda(queue &q, char n);
bool testa(queue &q, char &out);
bool estrai_testa(queue &q);
void stampa(const queue &q);

#endif
```

3 **queue.cc**

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

static int next(int index, const queue &q)
{
    return (index + 1) % q.dim;
}

void init(queue &q, int maxdim)
{
    q.tail = q.head = 0;
    q.dim = maxdim+1;
    q.elem = new char[q.dim];
}

void deinit(queue &q)
{
    delete[] q.elem;
}

static bool vuota(const queue &q)
{
    return q.tail == q.head;
}
```

```
static bool piena(const queue &q)
{
    return next(q.tail, q) == q.head;
}

bool accoda(queue &q, char n)
{
    bool ris = false;
    if (!piena(q)) {
      q.elem[q.tail] = n;
      q.tail = next(q.tail, q);
      ris = true;
    }
    return ris;
}

bool estrai_testa(queue &q)
{
    bool ris = false;
    if (!vuota(q)) {
      q.head = next(q.head, q);
      ris = true;
    }
    return ris;
}

bool testa(queue &q, char &out)
{
    bool ris = false;
    if (!vuota(q)) {
      out = q.elem[q.head];
      ris = true;
    }
    return ris;
}

void stampa(const queue &q)
{
    for (int i = q.head; i != q.tail; i = next(i, q)) {
        cout << q.elem[i] << endl;
    }
    cout << endl;
}
```

4 Completare il file "**esercizio4.cc**" definendo una funzione **RICORSIVA** `intersezione` che prenda in ingresso due **INSIEMI ORDINATI DI INTERI** rappresentati come array con dimensione virtuale e restituisca l'insieme **INTERSEZIONE** dei due insiemi come array con dimensione virtuale. Si assuma che gli array non contengano valori duplicati. E' vietato l'uso di cicli. E' permesso l'uso di funzioni wrapper.

(Per completare la main, si definiscano anche le funzioni `stampaarray` e `leggiarray` nel modo usuale, utilizzando cicli.)

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

2 **esercizio4.cc**

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

using namespace std;

void stampaarray(const int V[],int n);
void leggiarray(int V[],int & n);
void intersezione(const int A[], int dima,
                  const int B[], int dimb,
                  int C[], int & dimc) ;

int main () {
   int MAXDIM = 100;
   int A[MAXDIM] = {1,3,4,5,7,9,10,11,14,19,21,24};
   int dima = 12;
   int B[MAXDIM] = {0,1,2,4,5,6,8,11,12,13,15,19,22,23,24,27};
   int dimb = 16;
   int C[MAXDIM];
   int dimc;
   intersezione(A,dima,B,dimb,C,dimc);
   stampaarray(C,dimc);
   leggiarray(A,dima);
   leggiarray(B,dimb);
   intersezione(A,dima,B,dimb,C,dimc);
   stampaarray(C,dimc);
}

void leggiarray(int V[],int & n) {
   cout << "Dimensione: ";
   cin >> n;
   for (int i=0;i<n;i++) {
     cout << setw(2) << i << ": " ;
     cin >> V[i];
   }
}

void stampaarray(const int V[],int n) {
   cout << "[";
   for (int i=0;i<n;i++) {
     cout << setw(3) << V[i];
   }
   cout << "]" << endl;
}

void intersezione_ric(const int A[],int mina,int maxa,
                      const int B[],int minb,int maxb,
                      int C[], int & dimc) {
   if ((mina>maxa)||(minb>maxb))
     ;
   else if (A[mina]==B[minb]) {
     C[dimc]=A[mina];
     dimc++;
```

```
    intersezione_ric(A,mina+1,maxa,B,minb+1,maxb,C,dimc);
  }
  else if (A[mina]<B[minb]) {
    intersezione_ric(A,mina+1,maxa,B,minb,maxb,C,dimc);
  }
  else  { // (A[mina]>B[minb])
    intersezione_ric(A,mina,maxa,B,minb+1,maxb,C,dimc);
  }
}

void intersezione(const int A[], int dima,
                  const int B[],int dimb,
                  int C[], int & dimc) {
  dimc=0;
  intersezione_ric(A,0,dima-1,B,0,dimb-1,C,dimc);
}
```