Primo Appello di Programmazione I

08 Gennaio 2015
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 riceva in ingresso, come parametri da riga di comando, i nomi di due file di testo, rispettivamente di input e di output.

Il programma deve leggere il contenuto del file di input e scrivere, nel file di output, in ordine, il numero di occorrenze del carattere "`a`" in ciascuna parola, cosicché ad ogni parola del file in ingresso corrisponda un numero nel file in uscita.

Se ad esempio l'eseguibile è `a.out` ed il file `input.txt` ha il seguente contenuto:

```
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.
```

allora dopo l'esecuzione del comando:

```
./a.out input.txt output.txt
```

il file `output.txt` conterrà:

```
2 0 1 1 2 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 2 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 1 0
1 0 0 1 0 0 0 0 1
```

NOTA 1: Si assuma che nessuna parola del file di input contenga più di **255** caratteri.

NOTA 2: Il programma non deve prevedere alcun numero massimo di parole leggibili dal file di input, pena l'annullamento dell'esercizio.

NOTA 3: Non è consentito l'utilizzo di funzioni di libreria per gestire stringhe, in particolare della libreria `cstring` (come `strlen` o `strchr`).

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

1 **esercizio1.cc**

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

using namespace std;

const char CARATTERE = 'a';

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

  fstream my_in, my_out;

  char buf[256];

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

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

  // cin-loop
  while (my_in >> buf) {
    // Calcola numero di occorrenze di CARATTERE
    // nella parola
    int n = 0;
    // Non si puo' utilizzare strlen, ne' strchr,
    // ne' altre funzioni della libreria cstring
    for(int i = 0; buf[i] != '\0' && i < 256; i++) {
      if(buf[i] == CARATTERE) {
        n++;
      }
    }
    // Stampa nel file di output
    my_out << n << " ";
  }

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

Scrivere nel file `esercizio1.cc` un programma che riceva in ingresso, come parametri da riga di comando, i nomi di due file di testo, rispettivamente di input e di output.

Il programma deve leggere il contenuto del file di input e scrivere, nel file di output, in ordine, rispettivamente "1" se la parola comincia per **vocale** e "0" in tutti gli altri casi. Ad ogni parola del file in ingresso deve corrispondere un numero nel file in uscita.

Se ad esempio l'eseguibile è `a.out` ed il file `input.txt` ha il seguente contenuto:

1 `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.
```

allora dopo l'esecuzione del comando:

```
./a.out input.txt output.txt
```

il file `output.txt` conterrà:

```
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0 0
```

NOTA 1: Si assuma che nessuna parola del file di input contenga più di **255** caratteri.

NOTA 2: Il programma non deve prevedere alcun numero massimo di parole leggibili dal file di input, pena l'annullamento dell'esercizio.

NOTA 3: Non è consentito l'utilizzo di funzioni di libreria per gestire stringhe, in particolare della libreria `cstring` (come `strlen` o `strchr`).

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

1 **esercizio1.cc**

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

using namespace std;

// Vocali, minuscole e maiuscole
const int NUMERO_VOCALI = 10;
const char VOCALI[NUMERO_VOCALI + 1] = "aAeEiIoOuU";

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

  fstream my_in, my_out;

  char buf[256];

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

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

  // cin-loop
  while (my_in >> buf) {
    // Verifica se la parola comincia per vocale
    bool vocale = false;
    // Non si puo' utilizzare strlen, ne' strchr,
    // ne' altre funzioni della libreria cstring
    for(int i = 0; i < NUMERO_VOCALI && !vocale; i++) {
      if(buf[0] == VOCALI[i]) {
        vocale = true;
      }
    }
    // Stampa nel file di output
    if(vocale) {
      my_out << "1 ";
    } else {
      my_out << "0 ";
    }
  }

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

Scrivere nel file `esercizio1.cc` un programma che riceva in ingresso, come parametri da riga di comando, i nomi di due file di testo, rispettivamente di input e di output.

Il programma deve leggere il contenuto del file di input e scrivere, nel file di output, in ordine, la lunghezza di ciascuna parola letta, se essa è **maggiore od uguale a 5**, e **0** altrimenti,

cosicché ad ogni parola del file in ingresso corrisponda un numero nel file in uscita.

Se ad esempio l'eseguibile è `a.out` ed il file `input.txt` ha il seguente contenuto:

```
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.
```

allora dopo l'esecuzione del comando:

```
./a.out input.txt output.txt
```

il file `output.txt` conterrà:

```
11 5 7 6 7 0 0 6 0 6 0 0 5 6 0 6 0 6 0 0 9 0 0 6 0 0 6 0 7 0 5 5 7 6 5 0 6 6
6 0 0 5 0 8 0 0 7
```

NOTA 1: Si assuma che nessuna parola del file di input contenga più di **255** caratteri.

NOTA 2: Il programma non deve prevedere alcun numero massimo di parole leggibili dal file di input, pena l'annullamento dell'esercizio.

NOTA 3: Non è consentito l'utilizzo di funzioni di libreria per gestire stringhe, in particolare della libreria `cstring` (come `strlen` o `strchr`).

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

1 **esercizio1.cc**

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

using namespace std;

const int SOGLIA = 5;

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

  fstream my_in, my_out;

  char buf[256];

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

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

  // cin-loop
  while (my_in >> buf) {
    // Calcola la lunghezza della parola;
    // non si puo' utilizzare strlen, ne' strchr,
    // ne' altre funzioni della libreria cstring
    int lun;
    for(lun = 0; buf[lun] != '\0' && lun < 256; lun++);
    // Stampa nel file di output
    if(lun >= SOGLIA) {
      my_out << lun << " ";
    } else {
      my_out << "0 ";
    }
  }

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

Scrivere nel file `esercizio1.cc` un programma che riceva in ingresso, come parametri da riga di comando, i nomi di due file di testo, rispettivamente di input e di output.

Il programma deve leggere il contenuto del file di input e scrivere, nel file di output, in ordine, la lunghezza di ciascuna parola letta, se essa è **inferiore a 5**, e **0** altrimenti, cosicché ad ogni parola del file in ingresso corrisponda un numero nel file in uscita.

Se ad esempio l'eseguibile è `a.out` ed il file `input.txt` ha il seguente contenuto:

```
1 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.
```

allora dopo l'esecuzione del comando:

```
./a.out input.txt output.txt
```

il file `output.txt` conterrà:

```
0 0 0 0 0 3 2 0 2 0 2 2 0 0 3 0 2 0 1 2 0 2 4 0 3 3 0 4 0 4 0 0 0 0 0 1 0 0
0 3 4 0 2 0 2 2 0
```

NOTA 1: Si assuma che nessuna parola del file di input contenga più di **255** caratteri.

NOTA 2: Il programma non deve prevedere alcun numero massimo di parole leggibili dal file di input, pena l'annullamento dell'esercizio.

NOTA 3: Non è consentito l'utilizzo di funzioni di libreria per gestire stringhe, in particolare della libreria `cstring` (come `strlen` o `strchr`).

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

1 **esercizio1.cc**

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

using namespace std;

const int SOGLIA = 5;

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

  fstream my_in, my_out;

  char buf[256];

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

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

  // cin-loop
  while (my_in >> buf) {
    // Calcola la lunghezza della parola;
    // non si puo' utilizzare strlen, ne' strchr,
    // ne' altre funzioni della libreria cstring
    int lun;
    for(lun = 0; buf[lun] != '\0' && lun < 256; lun++);
    // Stampa nel file di output
    if(lun < SOGLIA) {
      my_out << lun << " ";
    } else {
      my_out << "0 ";
    }
  }

  my_in.close();
  my_out.close();
  return(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;
- 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 6 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 6 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 6 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 6 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`.

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 **soluzione_A31.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`.

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 **soluzione_A32.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`.

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 **soluzione_A33.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`.

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 **soluzione_A34.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 Nel file `esercizio4.cc` si realizzi una funzione **ricorsiva in coda** che implementi la funzione di Fibonacci `f(i)`.

NOTE:

(a) non sono ammessi cicli;

(b) le uniche funzioni ricorsive ammesse devono essere funzioni ricorsive in coda;

(c) è ammesso l'uso di funzioni wrapper.

Esempio di esecuzione:

```
./a.out
n? (n >= 0): 10
f( 0)=          1
f( 1)=          1
f( 2)=          2
f( 3)=          3
f( 4)=          5
f( 5)=          8
f( 6)=         13
f( 7)=         21
f( 8)=         34
f( 9)=         55
f(10)=         89
```

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

4 **soluzione_A41.cc**

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

long long f (int n);

int main()
{
  int n;
  cout << "n? (n >= 0): ";
  cin >> n;
  for (int i = 0;i <=n; i++)
    cout <<  "f(" << setw(2) << i << ")="
         << setw(10) <<  f(i) << endl;
return 0;
}

// SOLUZIONE:

long long f1 (long long fimenodue, long long fimenouno, int i, int n)
{
  long long fn;
  if (i==n)
    fn = fimenodue+fimenouno;
  else
    fn = f1(fimenouno,fimenouno+fimenodue,i+1,n);
  return fn;
}

long long f (int n) {
  long long fn;
  if (n==0||n==1)
    fn = 1;
  else
    fn = f1 (1,1,2,n);
  return fn;
}
```