Quarto Appello di Programmazione I

01 Luglio 2014
Prof. Roberto Sebastiani

Codice:

Nome	Cognome	Matricola

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

Modalità di questo appello

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

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

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

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

1 Scrivere nel file `esercizio1.cc` un programma che, dati due file di testo, crea un terzo file di testo in uscita contenente **tutte le parole che compaiono in entrambi i file in input**. Il programma deve cioè estrarre tutte le parole dal primo e dal secondo file e scrivere nel terzo solo quelle che compaiono almeno una volta in entrambi; i nomi dei due file di input e del file di output vengono letti come argomenti da riga di comando.

In questo contesto, una "parola" è una sequenza di caratteri separata da quella che la segue da uno spazio o da un a capo. Non è necessario eliminare eventuali parole duplicate; notare che lettere maiuscole e minuscole sono **diverse** (per esempio, "di" è diversa da "Di"). **Per semplicità si assuma che ogni file contenga al più 10.000 parole, e che ogni parola non contenga più di 100 caratteri.**

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: `primo file`

La costellazione giace lontano dalla scia luminosa della Via Lattea, dunque è priva di campi stellari di fondo; l'eclittica passa molto vicina all'estremità nordoccidentale della balena, perciò alcuni pianeti e la luna possono transitarvi brevemente.

Estratto da Wikipedia -
http://it.wikipedia.org/wiki/Balena_(costellazione)

Figura 2: `secondo file`

Esempio: se l'eseguibile è `a.out`, e i file di input quelli di Figura 1 e Figura 2, il comando

```
./a.out primo secondo output
```

creerà un file contenente le parole (ognuna su di una riga separata)

di di e la

NOTA 1: **Nel corso del programma non è ammesso aprire gli stream di accesso ai file più di una volta**; non è altresì ammesso utilizzare più di uno stream per file.

NOTA 2: È ammesso l'utilizzo delle funzioni `strcmp(const char *s1, const char *s2)` e `strcpy(char *s1, const char *s2)` della libreria `cstring`.

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 <cstring>

using namespace std;

const int MAX_PAROLE = 10000;
const int LUNGHEZZA_MAX_PAROLA = 100;

void inserisciSenzaDoppioni(char parola[LUNGHEZZA_MAX_PAROLA + 1],
                            char array[][LUNGHEZZA_MAX_PAROLA + 1],
                            int& lunghezzaArray);

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

  fstream primo, secondo, output;
  char tmp[LUNGHEZZA_MAX_PAROLA + 1];
  int parolePrimoFile = 0;
  int paroleIntersezione = 0;
  char parole[MAX_PAROLE][LUNGHEZZA_MAX_PAROLA + 1];
  char intersezione[MAX_PAROLE][LUNGHEZZA_MAX_PAROLA + 1];

  if (argc!=4) {
    cout << "Utilizzo: ./a.out <primo_file> <secondo_file> <file_di_output>\n";
    return -1;
  }

  primo.open(argv[1],ios::in);
  secondo.open(argv[2],ios::in);
  output.open(argv[3],ios::out);

  // Lettura primo file
  primo >> tmp;
  while(!primo.eof()) {
    // Inserimento della parola letta in fondo al primo array
    strcpy(parole[parolePrimoFile], tmp);
    // Aggiorna l'indice
    parolePrimoFile++;
    // Lettura prossima parola
    primo >> tmp;
  }

  // Lettura del secondo file e costruzione dell'insieme delle
  // parole comuni ai due file
  secondo >> tmp;
  while(!secondo.eof()) {
    // Ricerca della parola letta tra quelle del primo file
    int i = 0;
    bool trovata = false;
    while(i < parolePrimoFile && !trovata) {
      if(strcmp(tmp, parole[i]) == 0) {
        trovata = true;
```

```
      }
      i++;
    }
    if(trovata) {
      // Allora devo aggiungere una nuova parola
      // all'array contenente le parole comuni ai due file
      strcpy(intersezione[paroleIntersezione], tmp);
      // Aggiorna l'indice
      paroleIntersezione++;
    }
    // Lettura prossima parola
    secondo >> tmp;
  }

  // Scrittura sul file di output
  for(int i = 0; i < paroleIntersezione; i++) {
    output << intersezione[i] << endl;
  }

  primo.close();
  secondo.close();
  output.close();

  return (0);
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, dati due file di testo, crea un terzo file di testo in uscita contenente **le parole del primo file di input che non compaiono nel secondo**. Il programma deve cioè estrarre tutte le parole dal primo e dal secondo file e scrivere nel terzo solo quelle che compaiono esclusivamente nel primo; i nomi dei due file di input e del file di output vengono letti come argomenti da riga di comando.

In questo contesto, una "parola" è una sequenza di caratteri separata da quella che la segue da uno spazio o da un a capo. Non è necessario eliminare eventuali parole duplicate; notare che lettere maiuscole e minuscole sono **diverse** (per esempio, "di" è diversa da "Di"). **Per semplicità si assuma che ogni file contenga al più 10.000 parole, e che ogni parola non contenga più di 100 caratteri.**

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: `primo file`

La costellazione giace lontano dalla scia luminosa della Via Lattea, dunque è priva di campi stellari di fondo; l'eclittica passa molto vicina all'estremità nordoccidentale della balena, perciò alcuni pianeti e la luna possono transitarvi brevemente.

Estratto da Wikipedia -
http://it.wikipedia.org/wiki/Balena_(costellazione)

Figura 4: `secondo file`

Esempio: se l'eseguibile è `a.out`, e i file di input quelli di Figura 1 e Figura 2, il comando

```
./a.out primo secondo output
```

creerà un file contenente le parole (ognuna su di una riga separata)

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

NOTA 1: **Nel corso del programma non è ammesso aprire gli stream di accesso ai file più di una volta**; non è altresì ammesso utilizzare più di uno stream per file.

NOTA 2: È ammesso l'utilizzo delle funzioni `strcmp(const char *s1, const char *s2)` e `strcpy(char *s1, const char *s2)` della libreria `cstring`.

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 <cstring>

using namespace std;

const int MAX_PAROLE = 10000;
const int LUNGHEZZA_MAX_PAROLA = 100;

void inserisciSenzaDoppioni(char parola[LUNGHEZZA_MAX_PAROLA + 1],
                            char array[][LUNGHEZZA_MAX_PAROLA + 1],
                            int& lunghezzaArray);

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

  fstream primo, secondo, output;
  char tmp[LUNGHEZZA_MAX_PAROLA + 1];
  int paroleSecondoFile = 0;
  int paroleDifferenza = 0;
  char parole[MAX_PAROLE][LUNGHEZZA_MAX_PAROLA + 1];
  char differenza[MAX_PAROLE][LUNGHEZZA_MAX_PAROLA + 1];

  if (argc!=4) {
    cout << "Utilizzo: ./a.out <primo_file> <secondo_file> <file_di_output>\n";
    return -1;
  }

  primo.open(argv[1],ios::in);
  secondo.open(argv[2],ios::in);
  output.open(argv[3],ios::out);

  // Lettura secondo file
  secondo >> tmp;
  while(!secondo.eof()) {
    // Inserimento della parola letta in fondo al primo array
    strcpy(parole[paroleSecondoFile], tmp);
    // Aggiorna l'indice
    paroleSecondoFile++;
    // Lettura prossima parola
    secondo >> tmp;
  }

  // Lettura del primo file e costruzione dell'insieme delle
  // parole proprie solo del primo file
  primo >> tmp;
  while(!primo.eof()) {
    // Ricerca della parola letta tra quelle del secondo file
    int i = 0;
    bool trovata = false;
    while(i < paroleSecondoFile && !trovata) {
      if(strcmp(tmp, parole[i]) == 0) {
        trovata = true;
```

```
      }
      i++;
    }
    if(!trovata) {
      // Allora devo aggiungere una nuova parola
      // all'array contenente le parole proprie solo del primo file
      strcpy(differenza[paroleDifferenza], tmp);
      // Aggiorna l'indice
      paroleDifferenza++;
    }
    // Lettura prossima parola
    primo >> tmp;
  }

  // Scrittura sul file di output
  for(int i = 0; i < paroleDifferenza; i++) {
    output << differenza[i] << endl;
  }

  primo.close();
  secondo.close();
  output.close();

  return (0);
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, dati due file di testo, crea un terzo file di testo in uscita contenente **le parole del secondo file di input che non compaiono nel primo**. Il programma deve cioè estrarre tutte le parole dal primo e dal secondo file e scrivere nel terzo solo quelle che compaiono esclusivamente nel secondo; i nomi dei due file di input e del file di output vengono letti come argomenti da riga di comando.

In questo contesto, una "parola" è una sequenza di caratteri separata da quella che la segue da uno spazio o da un a capo. Non è necessario eliminare eventuali parole duplicate; notare che lettere maiuscole e minuscole sono **diverse** (per esempio, "di" è diversa da "Di"). **Per semplicità si assuma che ogni file contenga al più 10.000 parole, e che ogni parola non contenga più di 100 caratteri.**

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: `primo file`

La costellazione giace lontano dalla scia luminosa della Via Lattea, dunque è priva di campi stellari di fondo; l'eclittica passa molto vicina all'estremità nordoccidentale della balena, perciò alcuni pianeti e la luna possono transitarvi brevemente.

Estratto da Wikipedia - http://it.wikipedia.org/wiki/Balena_(costellazione)

Figura 6: `secondo file`

Esempio: se l'eseguibile è `a.out`, e i file di input quelli di Figura 1 e Figura 2, il comando

```
./a.out primo secondo output
```

creerà un file contenente le parole (ognuna su di una riga separata)

La costellazione giace lontano dalla scia luminosa della Via Lattea, dunque è priva campi stellari fondo; l'eclittica passa molto vicina all'estremità nordoccidentale della balena, perciò alcuni pianeti luna possono transitarvi brevemente. Estratto da Wikipedia - http://it.wikipedia.org/wiki/Balena_(costellazione)

NOTA 1: **Nel corso del programma non è ammesso aprire gli stream di accesso ai file più di una volta**; non è altresì ammesso utilizzare più di uno stream per file.

NOTA 2: È ammesso l'utilizzo delle funzioni `strcmp(const char *s1, const char *s2)` e `strcpy(char *s1, const char *s2)` della libreria `cstring`.

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 <cstring>

using namespace std;

const int MAX_PAROLE = 10000;
const int LUNGHEZZA_MAX_PAROLA = 100;

void inserisciSenzaDoppioni(char parola[LUNGHEZZA_MAX_PAROLA + 1],
                            char array[][LUNGHEZZA_MAX_PAROLA + 1],
                            int& lunghezzaArray);

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

  fstream primo, secondo, output;
  char tmp[LUNGHEZZA_MAX_PAROLA + 1];
  int parolePrimoFile = 0;
  int paroleDifferenza = 0;
  char parole[MAX_PAROLE][LUNGHEZZA_MAX_PAROLA + 1];
  char differenza[MAX_PAROLE][LUNGHEZZA_MAX_PAROLA + 1];

  if (argc!=4) {
    cout << "Utilizzo: ./a.out <primo_file> <secondo_file> <file_di_output>\n";
    return -1;
  }

  primo.open(argv[1],ios::in);
  secondo.open(argv[2],ios::in);
  output.open(argv[3],ios::out);

  // Lettura primo file
  primo >> tmp;
  while(!primo.eof()) {
    // Inserimento della parola letta in fondo al primo array
    strcpy(parole[parolePrimoFile], tmp);
    // Aggiorna l'indice
    parolePrimoFile++;
    // Lettura prossima parola
    primo >> tmp;
  }

  // Lettura del secondo file e costruzione dell'insieme delle
  // parole proprie solo del secondo file
  secondo >> tmp;
  while(!secondo.eof()) {
    // Ricerca della parola letta tra quelle del primo file
    int i = 0;
    bool trovata = false;
    while(i < parolePrimoFile && !trovata) {
      if(strcmp(tmp, parole[i]) == 0) {
        trovata = true;
```

```
      }
      i++;
    }
    if(!trovata) {
      // Allora devo aggiungere una nuova parola
      // all'array contenente le parole proprie solo del secondo file
      strcpy(differenza[paroleDifferenza], tmp);
      // Aggiorna l'indice
      paroleDifferenza++;
    }
    // Lettura prossima parola
    secondo >> tmp;
  }

  // Scrittura sul file di output
  for(int i = 0; i < paroleDifferenza; i++) {
    output << differenza[i] << endl;
  }

  primo.close();
  secondo.close();
  output.close();

  return (0);
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, dati due file di testo, crea un terzo file di testo in uscita contenente **tutte le parole che compaiono in entrambi i file in input**. Il programma deve cioè estrarre tutte le parole dal primo e dal secondo file e scrivere nel terzo solo quelle che compaiono almeno una volta in entrambi; i nomi dei due file di input e del file di output vengono letti come argomenti da riga di comando.

In questo contesto, una "parola" è una sequenza di caratteri separata da quella che la segue da uno spazio o da un a capo. Non è necessario eliminare eventuali parole duplicate; notare che lettere maiuscole e minuscole sono **diverse** (per esempio, "di" è diversa da "Di"). **Per semplicità si assuma che ogni file contenga al più 10.000 parole, e che ogni parola non contenga più di 100 caratteri.**

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: `primo file`

La costellazione giace lontano dalla scia luminosa della Via Lattea, dunque è priva di campi stellari di fondo; l'eclittica passa molto vicina all'estremità nordoccidentale della balena, perciò alcuni pianeti e la luna possono transitarvi brevemente.

Estratto da Wikipedia -
http://it.wikipedia.org/wiki/Balena_(costellazione)

Figura 8: `secondo file`

Esempio: se l'eseguibile è `a.out`, e i file di input quelli di Figura 1 e Figura 2, il comando

```
./a.out primo secondo output
```

creerà un file contenente le parole (ognuna su di una riga separata)

di di e la

NOTA 1: **Nel corso del programma non è ammesso aprire gli stream di accesso ai file più di una volta**; non è altresì ammesso utilizzare più di uno stream per file.

NOTA 2: È ammesso l'utilizzo delle funzioni `strcmp(const char *s1, const char *s2)` e `strcpy(char *s1, const char *s2)` della libreria `cstring`.

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 <cstring>

using namespace std;

const int MAX_PAROLE = 10000;
const int LUNGHEZZA_MAX_PAROLA = 100;

void inserisciSenzaDoppioni(char parola[LUNGHEZZA_MAX_PAROLA + 1],
                            char array[][LUNGHEZZA_MAX_PAROLA + 1],
                            int& lunghezzaArray);

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

  fstream primo, secondo, output;
  char tmp[LUNGHEZZA_MAX_PAROLA + 1];
  int parolePrimoFile = 0;
  int paroleIntersezione = 0;
  char parole[MAX_PAROLE][LUNGHEZZA_MAX_PAROLA + 1];
  char intersezione[MAX_PAROLE][LUNGHEZZA_MAX_PAROLA + 1];

  if (argc!=4) {
    cout << "Utilizzo: ./a.out <primo_file> <secondo_file> <file_di_output>\n";
    return -1;
  }

  primo.open(argv[1],ios::in);
  secondo.open(argv[2],ios::in);
  output.open(argv[3],ios::out);

  // Lettura primo file
  primo >> tmp;
  while(!primo.eof()) {
    // Inserimento della parola letta in fondo al primo array
    strcpy(parole[parolePrimoFile], tmp);
    // Aggiorna l'indice
    parolePrimoFile++;
    // Lettura prossima parola
    primo >> tmp;
  }

  // Lettura del secondo file e costruzione dell'insieme delle
  // parole comuni ai due file
  secondo >> tmp;
  while(!secondo.eof()) {
    // Ricerca della parola letta tra quelle del primo file
    int i = 0;
    bool trovata = false;
    while(i < parolePrimoFile && !trovata) {
      if(strcmp(tmp, parole[i]) == 0) {
        trovata = true;
```

```
      }
      i++;
    }
    if(trovata) {
      // Allora devo aggiungere una nuova parola
      // all'array contenente le parole comuni ai due file
      strcpy(intersezione[paroleIntersezione], tmp);
      // Aggiorna l'indice
      paroleIntersezione++;
    }
    // Lettura prossima parola
    secondo >> tmp;
  }

  // Scrittura sul file di output
  for(int i = 0; i < paroleIntersezione; i++) {
    output << intersezione[i] << endl;
  }

  primo.close();
  secondo.close();
  output.close();

  return (0);
}
```

2 Scrivere nel file `esercizio2.cc` una funzione **`gioca(...)`** che implementi la logica di un gioco di carte con le caratteristiche che seguono:

Lo scopo del gioco é totalizzare esattamente 21 punti sommando il valore di tutte carte estratte nel corso della partita. Se il punteggio totalizzato supera 21, il giocatore ha perso. Ad ogni turno il programma chiederá al giocatore se vuole proseguire nel gioco vedendo un'altra carta o se si vuole ritirare. Ad ogni turno il giocatore puó:

(a) aver totalizzato esattamente 21 punti, vincendo la partita.

(b) aver totalizzato piú di 21 punti, perdendo la partita.

(c) decidere di uscire dal gioco, accettando il punteggio corrente. Per uscire dal gioco il giocatore deve rispondere `no` alla richiesta di vedere un'altra carta.

(d) decidere di giocare un altro turno, rispondendo `si` alla richiesta di vedere un'altra carta.

Se il giocatore ha vinto o ha perso il gioco termina immediatamente ed il giocatore non puó richiedere altre carte.

NOTA 1: L'output del programma deve essere esattamente come riportato nell'esempio di esecuzione di seguito:

```
Il banco da carta con valore 4
    Hai totalizzato 4 punti
    Vuoi un'altra carta? (s/n) s
Il banco da carta con valore 4
    Hai totalizzato 8 punti
    Vuoi un'altra carta? (s/n) s
Il banco da carta con valore 2
    Hai totalizzato 10 punti
    Vuoi un'altra carta? (s/n) s
Il banco da carta con valore 9
    Hai totalizzato 19 punti
    Vuoi un'altra carta? (s/n) s
Il banco da carta con valore 3
    Hai totalizzato 22 punti
HAI PERSO, il tuo punteggio totale e' 22
```

NOTA 2: Per simulare l'estrazione casuale di una carta dal mazzo **va utilizzata** la funzione `pesca_carta()` definita in `mazzo.h` e `mazzo.o`.

NOTA 3: Non é consentito l'uso di variabili globali, `continue`, `return` o `break` per uscire dai cicli.

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

2 **soluzione_A21.cc**

```
#include <iostream>
#include "mazzo.h"      /* init_mazzo(), pesca_carta() */

using namespace std;

void gioca(int &punteggio);

int main() {
  int punteggio=0;
  init_mazzo();
  gioca(punteggio);
  if (punteggio == 21) {
    cout << "HAI VINTO!" << endl;
  } else {
    if (punteggio > 21)
      cout << "HAI PERSO, il tuo punteggio totale e' " << punteggio << endl;
    else
      cout << "TI SEI RITIRATO, il tuo punteggio totale e' " << punteggio << endl;
  }
}

void gioca(int &punteggio) {
  char vuole_carta;
  do {
    int carta = pesca_carta();
    cout << "Il banco da carta con valore " << carta << endl;
    punteggio += carta;
    cout << "    Hai totalizzato " << punteggio << " punti " << endl;
    if (punteggio < 21) {
      cout << "    Vuoi un'altra carta? (s/n) ";
      cin >> vuole_carta;
    }
  } while (punteggio < 21 && vuole_carta=='s');
}
```

2 Scrivere nel file `esercizio2.cc` una funzione **gioca(...)** che implementi la logica di un gioco di carte con le caratteristiche che seguono:

Lo scopo del gioco é totalizzare esattamente 15 punti sommando il valore di tutte carte estratte nel corso della partita. Se il punteggio totalizzato supera 15, il giocatore ha perso. A ogni turno il programma chiedera' al giocatore se vuole proseguire nel gioco vedendo un'altra carta o se si vuole ritirare. A ogni turno il giocatore puó:

(a) aver totalizzato esattamente 15 punti, vincendo la partita.

(b) aver totalizzato piú di 15 punti, perdendo la partita.

(c) decidere di uscire dal gioco, accettando il punteggio corrente. Per uscire dal gioco il giocatore deve rispondere `no` alla richiesta di vedere un'altra carta.

(d) decidere di giocare un altro turno, rispondendo `si` alla richiesta di vedere un'altra carta.

Se il giocatore ha vinto o ha perso il gioco termina immediatamente ed il giocatore non puó richiedere altre carte.

NOTA 1: L'output del programma deve essere esattamente come riportato nell'esempio di esecuzione di seguito:

```
Il banco da carta con valore 3
    Hai totalizzato 3 punti
    Vuoi un'altra carta? (s/n) s
Il banco da carta con valore 2
    Hai totalizzato 5 punti
    Vuoi un'altra carta? (s/n) s
Il banco da carta con valore 3
    Hai totalizzato 8 punti
    Vuoi un'altra carta? (s/n) s
Il banco da carta con valore 3
    Hai totalizzato 11 punti
    Vuoi un'altra carta? (s/n) s
Il banco da carta con valore 2
    Hai totalizzato 13 punti
    Vuoi un'altra carta? (s/n) s
Il banco da carta con valore 3
    Hai totalizzato 16 punti
HAI PERSO, il tuo punteggio totale e' 16
```

NOTA 2: Per simulare l'estrazione casuale di una carta dal mazzo **va utilizzata** la funzione `pesca_carta()` definita in `mazzo.h` e `mazzo.o`.

NOTA 3: Non é consentito l'uso di variabili globali, `continue`, `return` o `break` per uscire dai cicli.

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

2 **soluzione_A22.cc**

```
#include <iostream>
#include "mazzo.h"      /* init_mazzo(), pesca_carta() */

using namespace std;

void gioca(int &punteggio);

int main() {
  int punteggio=0;
  init_mazzo();
  gioca(punteggio);
  if (punteggio == 15) {
    cout << "HAI VINTO!" << endl;
  } else {
    if (punteggio > 15)
      cout << "HAI PERSO, il tuo punteggio totale e' " << punteggio << endl;
    else
      cout << "TI SEI RITIRATO, il tuo punteggio totale e' " << punteggio << endl;
  }
}

void gioca(int &punteggio) {
  char vuole_carta;
  do {
    int carta = pesca_carta();
    cout << "Il banco da carta con valore " << carta << endl;
    punteggio += carta;
    cout << "    Hai totalizzato " << punteggio << " punti " << endl;
    if (punteggio < 15) {
      cout << "    Vuoi un'altra carta? (s/n) ";
      cin >> vuole_carta;
    }
  } while (punteggio < 15 && vuole_carta=='s');
}
```

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

- `init` inizializzi la pila per contenere al più `dim` elementi;
- `deinit` liberi la memoria utilizzata dalla pila;
- `push` inserisca l'elemento passato come parametro nella pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `pop` tolga l'elemento in testa alla pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `top` legga l'elemento in testa alla pila 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 pila, dall'elemento più vecchio al più recente.

La pila deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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

3 stack_main.cc

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

int main ()
{
    char res;
    double val;
    stack s;

    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire la dimensione massima della pila: ";
        cin >> maxdim;
    }

    init(s, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << " Push (u)\n"
             << " Pop (o)\n"
             << " Top (t)\n"
             << " Print (p)\n"
             << " Quit (q)\n";
        cin >> res;
        switch (res) {
        case 'u':
            cout << "Val? : ";
            cin >> val;
            if (!push(s, val)) {
                cout << "Pila piena!\n";
            }
            break;
        case 'o':
            if (!pop(s)) {
                cout << "Pila vuota!\n";
            }
            break;
        case 't':
            if (!top(s, val)) {
                cout << "Pila vuota!\n";
            } else {
                cout << "Top = " << val << endl;
            }
            break;
        case 'p':
            print(s);
            break;
        case 'q':
            break;
        default:
            cout << "Opzione errata\n";
```

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

    deinit(s);

    return 0;
}
```

3 **stack.h**

```
#ifndef STRUCT_STACK_H
#define STRUCT_STACK_H

struct stack {
    int index;
    int dim;
    double *elem;
};

void init(stack &s, int maxdim);
void deinit(stack &s);
bool push(stack &s, double n);
bool top(const stack &s, double &out);
bool pop(stack &s);
void print(const stack &s);

#endif
```

3 **stack_.cc**

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

static bool is_empty(const stack &s)
{
    return s.index == 0;
}

static bool is_full(const stack &s)
{
    return s.index == s.dim;
}

void init(stack &s, int maxdim)
{
    s.index = 0;
    s.dim = maxdim;
    s.elem = new double[maxdim];
}

void deinit(stack &s)
{
    delete[] s.elem;
```

```
}

bool top(const stack &s, double &out)
{
    if (is_empty(s)) {
        return false;
    } else {
        out = s.elem[s.index-1];
        return true;
    }
}

bool push(stack &s, double n)
{
    if (is_full(s)) {
        return false;
    } else {
        s.elem[s.index++] = n;
        return true;
    }
}

bool pop(stack &s)
{
    if (is_empty(s)) {
        return false;
    } else {
        s.index--;
        return true;
    }
}

void print(const stack &s)
{
    for (int i = 0; i < s.index; i++) {
        cout << s.elem[i] << " ";
    }
    cout << endl;
}
```

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

- `init` inizializzi la pila per contenere al più `dim` elementi;
- `deinit` liberi la memoria utilizzata dalla pila;
- `push` inserisca l'elemento passato come parametro nella pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `pop` tolga l'elemento in testa alla pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `top` legga l'elemento in testa alla pila 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 pila, dall'elemento più vecchio al più recente.

La pila deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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

3 stack_main.cc

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

int main ()
{
    char res;
    char val;
    stack s;

    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire la dimensione massima della pila: ";
        cin >> maxdim;
    }

    init(s, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << " Push (u)\n"
             << " Pop (o)\n"
             << " Top (t)\n"
             << " Print (p)\n"
             << " Quit (q)\n";
        cin >> res;
        switch (res) {
        case 'u':
            cout << "Val? : ";
            cin >> val;
            if (!push(s, val)) {
                cout << "Pila piena!\n";
            }
            break;
        case 'o':
            if (!pop(s)) {
                cout << "Pila vuota!\n";
            }
            break;
        case 't':
            if (!top(s, val)) {
                cout << "Pila vuota!\n";
            } else {
                cout << "Top = " << val << endl;
            }
            break;
        case 'p':
            print(s);
            break;
        case 'q':
            break;
        default:
            cout << "Opzione errata\n";
```

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

    deinit(s);

    return 0;
}
```

3 **stack.h**

```
#ifndef STRUCT_STACK_H
#define STRUCT_STACK_H

struct stack {
    int index;
    int dim;
    char *elem;
};

void init(stack &s, int maxdim);
void deinit(stack &s);
bool push(stack &s, char n);
bool top(const stack &s, char &out);
bool pop(stack &s);
void print(const stack &s);

#endif
```

3 **stack_.cc**

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

static bool is_empty(const stack &s)
{
    return s.index == 0;
}

static bool is_full(const stack &s)
{
    return s.index == s.dim;
}

void init(stack &s, int maxdim)
{
    s.index = 0;
    s.dim = maxdim;
    s.elem = new char[maxdim];
}

void deinit(stack &s)
{
    delete[] s.elem;
```

```
}

bool top(const stack &s, char &out)
{
    if (is_empty(s)) {
        return false;
    } else {
        out = s.elem[s.index-1];
        return true;
    }
}

bool push(stack &s, char n)
{
    if (is_full(s)) {
        return false;
    } else {
        s.elem[s.index++] = n;
        return true;
    }
}

bool pop(stack &s)
{
    if (is_empty(s)) {
        return false;
    } else {
        s.index--;
        return true;
    }
}

void print(const stack &s)
{
    for (int i = 0; i < s.index; i++) {
        cout << s.elem[i] << " ";
    }
    cout << endl;
}
```

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

- `init` inizializzi la pila per contenere al più `dim` elementi;
- `deinit` liberi la memoria utilizzata dalla pila;
- `push` inserisca l'elemento passato come parametro nella pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `pop` tolga l'elemento in testa alla pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `top` legga l'elemento in testa alla pila 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 pila, dall'elemento più vecchio al più recente.

La pila deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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

3 stack_main.cc

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

int main ()
{
    char res;
    int val;
    stack s;

    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire la dimensione massima della pila: ";
        cin >> maxdim;
    }

    init(s, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << " Push (u)\n"
             << " Pop (o)\n"
             << " Top (t)\n"
             << " Print (p)\n"
             << " Quit (q)\n";
        cin >> res;
        switch (res) {
        case 'u':
            cout << "Val? : ";
            cin >> val;
            if (!push(s, val)) {
                cout << "Pila piena!\n";
            }
            break;
        case 'o':
            if (!pop(s)) {
                cout << "Pila vuota!\n";
            }
            break;
        case 't':
            if (!top(s, val)) {
                cout << "Pila vuota!\n";
            } else {
                cout << "Top = " << val << endl;
            }
            break;
        case 'p':
            print(s);
            break;
        case 'q':
            break;
        default:
            cout << "Opzione errata\n";
```

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

    deinit(s);

    return 0;
}
```

3 **stack.h**

```
#ifndef STRUCT_STACK_H
#define STRUCT_STACK_H

struct stack {
    int index;
    int dim;
    int *elem;
};

void init(stack &s, int maxdim);
void deinit(stack &s);
bool push(stack &s, int n);
bool top(const stack &s, int &out);
bool pop(stack &s);
void print(const stack &s);

#endif
```

3 **stack_.cc**

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

static bool is_empty(const stack &s)
{
    return s.index == 0;
}

static bool is_full(const stack &s)
{
    return s.index == s.dim;
}

void init(stack &s, int maxdim)
{
    s.index = 0;
    s.dim = maxdim;
    s.elem = new int[maxdim];
}

void deinit(stack &s)
{
    delete[] s.elem;
```

```
}

bool top(const stack &s, int &out)
{
    if (is_empty(s)) {
        return false;
    } else {
        out = s.elem[s.index-1];
        return true;
    }
}

bool push(stack &s, int n)
{
    if (is_full(s)) {
        return false;
    } else {
        s.elem[s.index++] = n;
        return true;
    }
}

bool pop(stack &s)
{
    if (is_empty(s)) {
        return false;
    } else {
        s.index--;
        return true;
    }
}

void print(const stack &s)
{
    for (int i = 0; i < s.index; i++) {
        cout << s.elem[i] << " ";
    }
    cout << endl;
}
```

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

- `init` inizializzi la pila per contenere al più `dim` elementi;
- `deinit` liberi la memoria utilizzata dalla pila;
- `push` inserisca l'elemento passato come parametro nella pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `pop` tolga l'elemento in testa alla pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `top` legga l'elemento in testa alla pila 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 pila, dall'elemento più vecchio al più recente.

La pila deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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

3 stack_main.cc

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

int main ()
{
    char res;
    float val;
    stack s;

    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire la dimensione massima della pila: ";
        cin >> maxdim;
    }

    init(s, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << " Push (u)\n"
             << " Pop (o)\n"
             << " Top (t)\n"
             << " Print (p)\n"
             << " Quit (q)\n";
        cin >> res;
        switch (res) {
        case 'u':
            cout << "Val? : ";
            cin >> val;
            if (!push(s, val)) {
                cout << "Pila piena!\n";
            }
            break;
        case 'o':
            if (!pop(s)) {
                cout << "Pila vuota!\n";
            }
            break;
        case 't':
            if (!top(s, val)) {
                cout << "Pila vuota!\n";
            } else {
                cout << "Top = " << val << endl;
            }
            break;
        case 'p':
            print(s);
            break;
        case 'q':
            break;
        default:
            cout << "Opzione errata\n";
```

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

    deinit(s);

    return 0;
}
```

3 **stack.h**

```
#ifndef STRUCT_STACK_H
#define STRUCT_STACK_H

struct stack {
    int index;
    int dim;
    float *elem;
};

void init(stack &s, int maxdim);
void deinit(stack &s);
bool push(stack &s, float n);
bool top(const stack &s, float &out);
bool pop(stack &s);
void print(const stack &s);

#endif
```

3 **stack_.cc**

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

static bool is_empty(const stack &s)
{
    return s.index == 0;
}

static bool is_full(const stack &s)
{
    return s.index == s.dim;
}

void init(stack &s, int maxdim)
{
    s.index = 0;
    s.dim = maxdim;
    s.elem = new float[maxdim];
}

void deinit(stack &s)
{
    delete[] s.elem;
```

```
}

bool top(const stack &s, float &out)
{
    if (is_empty(s)) {
        return false;
    } else {
        out = s.elem[s.index-1];
        return true;
    }
}

bool push(stack &s, float n)
{
    if (is_full(s)) {
        return false;
    } else {
        s.elem[s.index++] = n;
        return true;
    }
}

bool pop(stack &s)
{
    if (is_empty(s)) {
        return false;
    } else {
        s.index--;
        return true;
    }
}

void print(const stack &s)
{
    for (int i = 0; i < s.index; i++) {
        cout << s.elem[i] << " ";
    }
    cout << endl;
}
```

4 L'algoritmo "Insertsort" per ordinare un'array **A** di dimensione **dim** in ordine crescente funziona come segue:

- per $m = 1, ..., dim - 1$, all'm-sima iterazione viene ordinato il sottoarray $A[0..m]$, inserendo nella giusta posizione l'elemento $A[m]$ nell'array $A[0..m-1]$ precedentemente ordinato, e spostando in avanti di una posizione gli elementi ad esso maggiori;
- per inserire nella giusta posizione l'elemento $A[m]$ nell'array $A[0..m-1]$, si scandisce l'array $A[0..m-1]$ per $j = m-1, ..., 0$ fintanto che $A[j] > A[j+1]$, scambiando $A[j]$ e $A[j+1]$.

Si realizzi questo algoritmo in forma COMPLETAMENTE RICORSIVA.

NOTA 1: Non è consentito utilizzare alcuna forma di ciclo, `break`, `continue`, `goto`, né l'uso di variabili globali o di funzioni di libreria.

NOTA 2: È consentito definire e utilizzare eventuali funzioni ausiliarie, purché a loro volta ricorsive e senza cicli.

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

4 **codice_A41.cc**

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

void rec_insertsort(int [], int);
void readarray(int A[], int & dim);
void printarray(int A[], int min, int max);

const int MAXDIM = 100;

int main () {
  // Esempi
  int myarray[MAXDIM] = {
  41,3,9,1,5,17,6,20,37,2,8,23,10,0,11,19
//   1,3,4,5,2,6,7,8,9,10,11,12,13,14,15,16
//  10,3,9,1,5,17,6,41,20,37,2,8,23,0,11,19
//  11,12,13,14,15,16,1,2,3,4,5,6,7,8,9,10
//   1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
    };

  int dim = 16;
  printarray(myarray, 0, dim-1);
  cout << endl;
  rec_insertsort(myarray, dim);
  printarray(myarray, 0, dim-1);

  readarray(myarray, dim);
  printarray(myarray, 0, dim-1);
  cout << endl;
  rec_insertsort(myarray, dim);
  printarray(myarray, 0, dim-1);
}

// Legge l'array A[0...dim-1], dim <= MAXDIM
void readarray(int A[], int & dim) {
  cout << "dimensione array? ";
  cin >> dim;
  for (int i=0; i<dim; i++) {
    cout << "A[" << i << "]: ";
    cin >> A[i];
  }
}

// Stampa il subarray A[min...max], 0 <= min <= max <= MAXDIM
void printarray(int A[], int min, int max) {
  int i;
  cout << "[";
  for (i = min; i <= max; i++) {
    cout << setw(3) << A[i] << " ";
  }
  cout << "]\n";
}
```

```
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// %%% inserire qui sotto la definizione di rec_insertsort e
// %%% delle eventuali funzioni ausiliarie
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4 **soluzione_A41.cc**

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

void rec_insertsort(int [], int);
void readarray(int A[], int & dim);
void printarray(int A[], int min, int max);

const int MAXDIM = 100;

int main () {
  // Esempi
  int myarray[MAXDIM] = {
  41,3,9,1,5,17,6,20,37,2,8,23,10,0,11,19
//   1,3,4,5,2,6,7,8,9,10,11,12,13,14,15,16
//  10,3,9,1,5,17,6,41,20,37,2,8,23,0,11,19
//  11,12,13,14,15,16,1,2,3,4,5,6,7,8,9,10
//   1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
    };

  int dim = 16;
  printarray(myarray, 0, dim-1);
  cout << endl;
  rec_insertsort(myarray, dim);
  printarray(myarray, 0, dim-1);

  readarray(myarray, dim);
  printarray(myarray, 0, dim-1);
  cout << endl;
  rec_insertsort(myarray, dim);
  printarray(myarray, 0, dim-1);
}

// Legge l'array A[0...dim-1], dim <= MAXDIM
void readarray(int A[], int & dim) {
  cout << "dimensione array? ";
  cin >> dim;
  for (int i=0; i<dim; i++) {
    cout << "A[" << i << "]: ";
    cin >> A[i];
  }
}

// Stampa il subarray A[min...max], 0 <= min <= max <= MAXDIM
void printarray(int A[], int min, int max) {
```

```
  int i;
  cout << "[";
  for (i = min; i <= max; i++) {
    cout << setw(3) << A[i] << " ";
  }
  cout << "]\n";
}

// Inverte due elementi
void swap (int & a, int & b) {
  int c = a;
  a = b;
  b = c;
}

// Assumendo che il sub-array A[0..m-1] sia gia' ordinato,
// ordina ricorsivamente A[0..m] inserendo A[m] al posto giusto
// e spostando in avanti gli elementi successivi.
void rec_insert(int A [], int m) {
  if ((m > 0) && A[m] < A[m-1]) {
    swap(A[m], A[m-1]);
    rec_insert(A, m-1);
  }
}

// ordina ricorsivamente il sub-array A[0..m-1]
void rec_insertsort(int A[], int m) {
  if (m > 1) {
    rec_insertsort(A, m-1);
    rec_insert(A, m-1);
  }
}
```