Terzo Appello di Programmazione I

05 Giugno 2013
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 contenenti un numero indefinito di parole, generi un terzo file contenente nell'ordine la prima parola del **primo file**, la prima parola del **secondo file**, la seconda parola del primo file, la seconda parola del secondo file e così via, fino a copiare tutte le parole dei due file specificati in input. Quando uno dei due file termina, le parole dell'altro devono venire inserite in sequenza.

I nomi dei file di testo in input e del nuovo file su cui effettuare l'output sono passati al programma, nell'ordine, da linea di comando.

Dati, ad esempio, in input il file `testo1` contenente i seguenti dati:

```
Filastrocca delle parole: Fatevi avanti! Chi ne vuole?
```

ed il file `testo2` contenente i seguenti dati:

```
a bb ccc dddd eeeee ffffff
```

se l'eseguibile è `a.out`, il comando

```
./a.out testo1 testo2 output
```

genererà un file chiamato:

```
output
```

il cui contenuto è:

```
Filastrocca a delle bb parole: ccc Fatevi dddd avanti! eeeee Chi ffffff ne vuole?
```

NOTA: per semplicità considerare come *parola* una qualsiasi serie di caratteri compresi tra due spazi bianchi (e/o separatori di tabulazione, nuova linea e fine file). Con questa definizione anche strighe del tipo `parole:` o `vuole?` sono da considerarsi parole.

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

1 **soluzione_A11.cc**

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

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

  fstream my_in1, my_in2, my_out;
  char tmp1[100], tmp2[100];

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile1> <sourcefile2> <newfile>\n";
    exit(0);
  }

  my_in1.open(argv[1],ios::in);
  my_in2.open(argv[2],ios::in);
  my_out.open(argv[3],ios::out);

  my_in1 >> tmp1; //lettura prima parola dal primo file
  my_in2 >> tmp2; //lettura prima parola dal secondo file

  while ((!my_in1.eof()) && (!my_in2.eof())) {
    my_out << tmp1 << " " << tmp2 <<" ";
    my_in1 >> tmp1;
    my_in2 >> tmp2;
  }

  if (my_in1.eof()) {  //condizione per cui il primo file e' piu' corto del secondo
    while (!my_in2.eof()) {
      my_out << tmp2 <<" ";
      my_in2 >> tmp2;
    }
  }

  if (my_in2.eof()) {  //condizione per cui il secondo file e' piu' corto del primo
    while (!my_in1.eof()) {
      my_out << tmp1 <<" ";
      my_in1 >> tmp1;
    }
  }

  my_in1.close();
  my_in2.close();
  my_out.close();
  return(0);
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, dati due file di testo contenenti un numero indefinito di parole, generi un terzo file contenente nell'ordine la prima parola del **secondo file**, la prima parola del **primo file**, la seconda parola del secondo file, la seconda parola del primo file e così via, fino a copiare tutte le parole dei due file specificati in input. Quando uno dei due file termina, le parole dell'altro devono venire inserite in sequenza.

I nomi dei file di testo in input e del nuovo file su cui effettuare l'output sono passati al programma, nell'ordine, da linea di comando.

Dati, ad esempio, in input il file `testo1` contenente i seguenti dati:

```
Filastrocca delle parole: Fatevi avanti! Chi ne vuole?
```

ed il file `testo2` contenente i seguenti dati:

```
a bb ccc dddd eeeee ffffff
```

se l'eseguibile è `a.out`, il comando

```
./a.out testo1 testo2 output
```

genererà un file chiamato:

```
output
```

il cui contenuto è:

```
a Filastrocca bb delle ccc parole: dddd Fatevi eeeee avanti! ffffff Chi ne vuole?
```

NOTA: per semplicità considerare come *parola* una qualsiasi serie di caratteri compresi tra due spazi bianchi (e/o separatori di tabulazione, nuova linea e fine file). Con questa definizione anche strighe del tipo `parole:` o `vuole?` sono da considerarsi parole.

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

1 **soluzione_A12.cc**

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

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

  fstream my_in1, my_in2, my_out;
  char tmp1[100], tmp2[100];

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile1> <sourcefile2> <newfile>\n";
    exit(0);
  }

  my_in1.open(argv[1],ios::in);
  my_in2.open(argv[2],ios::in);
  my_out.open(argv[3],ios::out);

  my_in1 >> tmp1; //lettura prima parola dal primo file
  my_in2 >> tmp2; //lettura prima parola dal secondo file

  while ((!my_in1.eof()) && (!my_in2.eof())) {
    my_out << tmp2 << " " << tmp1 <<" ";
    my_in1 >> tmp1;
    my_in2 >> tmp2;
  }

  if (my_in1.eof()) {  //condizione per cui il primo file e' piu' corto del secondo
    while (!my_in2.eof()) {
      my_out << tmp2 <<" ";
      my_in2 >> tmp2;
    }
  }

  if (my_in2.eof()) {  //condizione per cui il secondo file e' piu' corto del primo
    while (!my_in1.eof()) {
      my_out << tmp1 <<" ";
      my_in1 >> tmp1;
    }
  }

  my_in1.close();
  my_in2.close();
  my_out.close();
  return(0);
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, dati due file di testo contenenti un numero indefinito di numeri interi, generi un terzo file contenente nell'ordine il primo numero del **primo file**, il primo numero del **secondo file**, il secondo numero del primo file, il secondo numero del secondo file e così via, fino a copiare tutti i numeri dei due file specificati in input. Quando uno dei due file termina, i numeri dell'altro devono venire inseriti in sequenza.

I nomi dei file di testo in input e del nuovo file su cui effettuare l'output sono passati al programma, nell'ordine, da linea di comando.

Dati, ad esempio, in input il file `testo1` contenente i seguenti dati:

```
1 1 2 3 5 8 13 21 34 55
```

ed il file `testo2` contenente i seguenti dati:

```
0 1 2 3 4 5 6 7 8 9 10 11 12
```

se l'eseguibile è `a.out`, il comando

```
./a.out testo1 testo2 output
```

genererà un file chiamato:

```
output
```

il cui contenuto è:

```
1 0 1 1 2 2 3 3 5 4 8 5 13 6 21 7 34 8 55 9 10 11 12
```

NOTA: consideriamo come *intero* qualsiasi sequenza di cifre decimali separate da spazi, eventualmente preceduta da un segno meno (-): numeri scritti in forma non canonica nei file di origine verranno salvati in forma canonica (ad esempio, "031" diventerà "31" nel file di output).

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

1 **soluzione_A13.cc**

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

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

  fstream my_in1, my_in2, my_out;
  int tmp1, tmp2;

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile1> <sourcefile2> <newfile>\n";
    exit(0);
  }

  my_in1.open(argv[1],ios::in);
  my_in2.open(argv[2],ios::in);
  my_out.open(argv[3],ios::out);

  my_in1 >> tmp1; //lettura primo numero dal primo file
  my_in2 >> tmp2; //lettura primo numero dal secondo file

  while ((!my_in1.eof()) && (!my_in2.eof())) {
    my_out << tmp1 << " " << tmp2 <<" ";
    my_in1 >> tmp1;
    my_in2 >> tmp2;
  }

  if (my_in1.eof()) {  //condizione per cui il primo file e' piu' corto del secondo
    while (!my_in2.eof()) {
      my_out << tmp2 <<" ";
      my_in2 >> tmp2;
    }
  }

  if (my_in2.eof()) {  //condizione per cui il secondo file e' piu' corto del primo
    while (!my_in1.eof()) {
      my_out << tmp1 <<" ";
      my_in1 >> tmp1;
    }
  }

  my_in1.close();
  my_in2.close();
  my_out.close();
  return(0);
}
```

1 Scrivere nel file `esercizio1.cc` un programma che, dati due file di testo contenenti un numero indefinito di numeri interi, generi un terzo file contenente nell'ordine il primo numero del **secondo file**, il primo numero del **primo file**, il secondo numero del secondo file, il secondo numero del primo file e così via, fino a copiare tutti i numeri dei due file specificati in input. Quando uno dei due file termina, i numeri dell'altro devono venire inseriti in sequenza.

I nomi dei file di testo in input e del nuovo file su cui effettuare l'output sono passati al programma, nell'ordine, da linea di comando.

Dati, ad esempio, in input il file `testo1` contenente i seguenti dati:

```
1 1 2 3 5 8 13 21 34 55
```

ed il file `testo2` contenente i seguenti dati:

```
0 1 2 3 4 5 6 7 8 9 10 11 12
```

se l'eseguibile è `a.out`, il comando

```
./a.out testo1 testo2 output
```

genererà un file chiamato:

```
output
```

il cui contenuto è:

```
0 1 1 1 2 2 3 3 4 5 5 8 6 13 7 21 8 34 9 55 10 11 12
```

NOTA: consideriamo come *intero* qualsiasi sequenza di cifre decimali separate da spazi, eventualmente preceduta da un segno meno (-): numeri scritti in forma non canonica nei file di origine verranno salvati in forma canonica (ad esempio, "031" diventerà "31" nel file di output).

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

1 **soluzione_A14.cc**

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

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

  fstream my_in1, my_in2, my_out;
  int tmp1, tmp2;

  if (argc!=4) {
    cout << "Usage: ./a.out <sourcefile1> <sourcefile2> <newfile>\n";
    exit(0);
  }

  my_in1.open(argv[1],ios::in);
  my_in2.open(argv[2],ios::in);
  my_out.open(argv[3],ios::out);

  my_in1 >> tmp1; //lettura primo numero dal primo file
  my_in2 >> tmp2; //lettura primo numero dal secondo file

  while ((!my_in1.eof()) && (!my_in2.eof())) {
    my_out << tmp2 << " " << tmp1 <<" ";
    my_in1 >> tmp1;
    my_in2 >> tmp2;
  }

  if (my_in1.eof()) {  //condizione per cui il primo file e' piu' corto del secondo
    while (!my_in2.eof()) {
      my_out << tmp2 <<" ";
      my_in2 >> tmp2;
    }
  }

  if (my_in2.eof()) {  //condizione per cui il secondo file e' piu' corto del primo
    while (!my_in1.eof()) {
      my_out << tmp1 <<" ";
      my_in1 >> tmp1;
    }
  }

  my_in1.close();
  my_in2.close();
  my_out.close();
  return(0);
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `crea_vettore` che, dato un intero positivo `N`, crei un nuovo vettore contenente i primi `N` quadrati perfetti, cominciando dallo zero.

Per esempio, i primi 6 quadrati perfetti sono $0, 1, 4, 9, 16, 25$ (i.e. rispettivamente $0^2, 1^2, 2^2, 3^2, 4^2, 5^2$).

NOTA 1: **La funzione `crea_vettore` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **codice_A21.cc**

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

// Inserire qui sotto la dichiarazione della funzione crea_vettore



int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;

   if (N < 0) {
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vector = crea_vettore(N);

   cout << "Array creato: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore
```

2 **soluzione_A21.cc**

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

// Inserire qui sotto la dichiarazione della funzione crea_vettore

int* crea_vettore(int n);
void quadrati_perfetti(int v[], int n, int i);

int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;
```

```
   if (N < 0) {
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vector = crea_vettore(N);

   cout << "Array creato: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

int* crea_vettore(int n) {
   int *v = new int[n];
   quadrati_perfetti(v, n, 0);
   return v;
}

void quadrati_perfetti(int v[], int n, int i) {
   if (i>= n) {
      return;
   } else {
      v[i] = i*i;
      quadrati_perfetti(v, n, i+1);
   }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `crea_vettore` che, dato un intero positivo `N`, crei un nuovo vettore contenente i primi `N` numeri triangolari, partendo da zero.

L'i-esimo numero triangolare si ottiene con la formula di Gauss

$$T(i) = \frac{i(i+1)}{2}.$$

Per esempio, i primi 8 numeri triangolari sono $0, 1, 3, 6, 10, 15, 21, 28$.

NOTA 1: **La funzione `crea_vettore` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **codice_A22.cc**

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

// Inserire qui sotto la dichiarazione della funzione crea_vettore


int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;

   if (N < 0) {
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vector = crea_vettore(N);

   cout << "Array creato: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore
```

2 **soluzione_A22.cc**

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

// Inserire qui sotto la dichiarazione della funzione crea_vettore

int* crea_vettore(int n);
void numeri_triangolari(int v[], int n, int i);

int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;

   if (N < 0) {
```

```
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vector = crea_vettore(N);

   cout << "Array creato: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

int* crea_vettore(int n) {
   int *v = new int[n];
   numeri_triangolari(v, n, 0);
   return v;
}

void numeri_triangolari(int v[], int n, int i) {
   if (i>= n) {
      return;
   } else {
      v[i] = (i*(i+1))/2;
      numeri_triangolari(v, n, i+1);
   }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `crea_vettore` che, dato un intero positivo `N`, crei un nuovo vettore contenente i primi `N` numeri esagonali, partendo da zero.

L'i-esimo numero esagonale si ottiene con la formula $E(i) = i(2i - 1)$.

Per esempio, i primi 7 numeri esagonali sono $0, 1, 6, 15, 28, 45, 66$.

NOTA 1: **La funzione `crea_vettore` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **codice_A23.cc**

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

// Inserire qui sotto la dichiarazione della funzione crea_vettore


int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;

   if (N < 0) {
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vector = crea_vettore(N);

   cout << "Array creato: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore
```

2 **soluzione_A23.cc**

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

// Inserire qui sotto la dichiarazione della funzione crea_vettore

int* crea_vettore(int n);
void numeri_esagonali(int v[], int n, int i);

int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;

   if (N < 0) {
```

```
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vector = crea_vettore(N);

   cout << "Array creato: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

int* crea_vettore(int n) {
   int *v = new int[n];
   numeri_esagonali(v, n, 0);
   return v;
}

void numeri_esagonali(int v[], int n, int i) {
   if (i>= n) {
      return;
   } else {
      v[i] = i*(2*i-1);
      numeri_esagonali(v, n, i+1);
   }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `crea_vettore` che, dato un intero positivo `N`, crei un nuovo vettore contenente i primi `N` numeri pentagonali, partendo da zero.

L'i-esimo numero pentagonale si ottiene con la seguente formula

$$P(i) = \frac{i(3i-1)}{2}.$$

Per esempio, i primi 9 numeri pentagonali sono $0, 1, 5, 12, 22, 35, 51, 70, 92$.

NOTA 1: **La funzione `crea_vettore` deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

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

2 **codice_A24.cc**

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

// Inserire qui sotto la dichiarazione della funzione crea_vettore


int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;

   if (N < 0) {
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vector = crea_vettore(N);

   cout << "Array creato: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore
```

2 **soluzione_A24.cc**

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

// Inserire qui sotto la dichiarazione della funzione crea_vettore

int* crea_vettore(int n);
void numeri_pentagonali(int v[], int n, int i);

int main(){

   int N;

   cout << "Dimensione: ";
   cin >> N;
```

```
   if (N < 0) {
      cout << "Attenzione: inserire intero positivo!\n";
      return 1;
   }

   int* vector = crea_vettore(N);

   cout << "Array creato: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   delete[] vector;

return 0;
}

// Inserire qui sotto la definizione della funzione crea_vettore

int* crea_vettore(int n) {
   int *v = new int[n];
   numeri_pentagonali(v, n, 0);
   return v;
}

void numeri_pentagonali(int v[], int n, int i) {
   if (i>= n) {
      return;
   } else {
      v[i] = (i*(3*i-1))/2;
      numeri_pentagonali(v, n, i+1);
   }
}
```

3 Nel file `collezione_main.cc` è definita la funzione `main` che contiene un menù per gestire un array ordinato di puntatori a una **struct** chiamata "elemento". L'array ha dimensione massima prefissata. Ogni "elemento" contiene un valore `int` chiamato "val". Scrivere, in un nuovo file `collezione.cc`, le definizioni delle funzioni dichiarate nello header file `collezione.h` in modo tale che:

- `init` inizializzi la collezione;
- `empty` controlli se la collezione è vuota, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `full` controlli se la collezione è piena, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `insert` inserisca l'elemento passato come parametro nella collezione, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti. La collezione deve essere ordinata in maniera **crescente** e se l'elemento è già presente deve essere inserito ugualmente. Esempio: l'inserimento dei seguenti valori:

  ```
  3  7  1  5  3
  ```

  deve produrre il seguente array:

  ```
  1  3  3  5  7
  ```

- `search` cerchi nella collezione l'elemento passato in input, restituendo `TRUE` se l'elemento è presente, e `FALSE` altrimenti: la funzione deve richiedere un numero di passi al più logaritmico rispetto al numero di elementi effettivamente contenuti nell'array;
- `print` stampi a video il contenuto della collezione, in ordine **crescente**.

NOTA: non è ammesso alterare in alcun modo i file `collezione_main.cc` e `collezione.h`.

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

3 collezione_A31_main.cc

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

int  main () {
  char res;
  collezione c;
  int val;
  init (c);
  do {
    cout << "\nOperazioni possibili:\n"
      << " Inserimento (i)\n"
      << " Ricerca (r)\n"
      << " Stampa ordinata (s)\n"
      << " Fine (f)\n";
    cout << "Operazione selezionata: ";
    cin >> res;
    switch (res) {
    case 'i':
      cout << "Valore : ";
      cin >> val;
      if (insert (c, val) == FALSE) {
cout << "Array pieno!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (search (c, val) == TRUE) {
cout << "Valore presente: " << val << endl;
      } else {
cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (empty (c) == TRUE) {
cout << "Array vuoto!" << endl;
      } else {
print (c);
      }
      break;
    case 'f':
      break;
    default:
      cout << "Opzione errata\n";
    }
  } while (res != 'f');

  return 0;
}
```

3 collezione_A31.h

```
// -*-C++ - *-
#define MAX_ARRAY_LENGTH 100

struct elemento {
  int val;
};

struct array {
  int dim;
  elemento *elementi[MAX_ARRAY_LENGTH];
};

typedef array *collezione;

enum boolean {
  FALSE, TRUE
};

void init (collezione &c);
boolean empty (const collezione &c);
boolean full (const collezione &c);
boolean insert (collezione &c, int val);
boolean search (const collezione &c, int val);
void print (const collezione &c);
```

3 soluzione_A31.cc

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

using namespace std;

void init (collezione &c) {
  // Inizializza la collezione
  c = new array;
  c->dim = 0;
}

boolean empty (const collezione &c) {
  // Ritorna "TRUE" se l'array  vuoto, "FALSE" altrimenti
  return (c->dim == 0) ? TRUE : FALSE;
}

boolean full (const collezione &c) {
  // Ritorna "TRUE" se l'array  pieno, "FALSE" altrimenti
  return (c->dim == MAX_ARRAY_LENGTH) ? TRUE : FALSE;
}

boolean insert (collezione &c, int val) {
  // Inserimento ordinato di un elemento (si assume l'array gi ordinato
  // in senso CRESCENTE)
  if (full (c) == FALSE) {
    int indice = 0;
    if (empty (c) == FALSE) {
```

```
      while (indice < c->dim && val > c->elementi[indice]->val) {
        indice++;
      }
      // Sposta il contenuto dell'array a destra di una posizione
      for (int i = c->dim; i > indice; i--) {
        c->elementi[i] = c->elementi[i - 1];
      }
    }
    // Aggiunge una posizione all'array
    c->dim++;
    // Inserisce il contenuto nell'array
    elemento *e = new elemento;
    e->val = val;
    // L'elemento va inserito in posizione "i"
    c->elementi[indice] = e;
    return TRUE;
  }
  return FALSE;
}

boolean search_rec (const collezione &c, int val, int start, int end) {
  if (start > end) {
    return FALSE;
  }
  // Trovo il "centro" dell'array
  int center = start + (end - start) / 2;
  if (val == c->elementi[center]->val) {
    return TRUE;
  } else if (val < c->elementi[center]->val) {
    // Prendo il lato sinistro dell'array
    return search_rec (c, val, start, center - 1);
  } else {
    // Prendo il lato destro dell'array
    return search_rec (c, val, center + 1, end);
  }
}

boolean search (const collezione &c, int val) {
  if (empty (c) == TRUE) {
    return FALSE;
  } else {
    return search_rec (c, val, 0, c->dim - 1);
  }
}

void print (const collezione &c) {
  if (empty (c) == FALSE) {
    // Stampo tutti gli elementi dell'array
    cout << "Array: [";
    for (int i = 0; i < c->dim; i++) {
      cout << c->elementi[i]->val << " ";
    }
    cout << "]" << endl;
  } else {
```

```
    cout << "Array vuoto." << endl;
  }
}
```

3 Nel file `collezione_main.cc` è definita la funzione `main` che contiene un menù per gestire un array ordinato di puntatori a una **struct** chiamata "elemento". L'array ha dimensione massima prefissata. Ogni "elemento" contiene un valore `long` chiamato "val". Scrivere, in un nuovo file `collezione.cc`, le definizioni delle funzioni dichiarate nello header file `collezione.h` in modo tale che:

- `init` inizializzi la collezione;
- `empty` controlli se la collezione è vuota, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `full` controlli se la collezione è piena, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `insert` inserisca l'elemento passato come parametro nella collezione, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti. La collezione deve essere ordinata in maniera **decrescente** e se l'elemento è già presente deve essere inserito ugualmente. Esempio: l'inserimento dei seguenti valori:

  ```
  3000000  7000000  10000000  5000000  3000000
  ```

  deve produrre il seguente array:

  ```
  10000000  7000000  5000000  3000000  3000000
  ```

- `search` cerchi nella collezione l'elemento passato in input, restituendo `TRUE` se l'elemento è presente, e `FALSE` altrimenti: la funzione deve richiedere un numero di passi al più logaritmico rispetto al numero di elementi effettivamente contenuti nell'array;
- `print` stampi a video il contenuto della collezione, in ordine **decrescente**.

NOTA: non è ammesso alterare in alcun modo i file `collezione_main.cc` e `collezione.h`.

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

3 collezione_A32_main.cc

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

int  main () {
  char res;
  collezione c;
  long val;
  init (c);
  do {
    cout << "\nOperazioni possibili:\n"
      << " Inserimento (i)\n"
      << " Ricerca (r)\n"
      << " Stampa ordinata (s)\n"
      << " Fine (f)\n";
    cout << "Operazione selezionata: ";
    cin >> res;
    switch (res) {
    case 'i':
      cout << "Valore : ";
      cin >> val;
      if (insert (c, val) == FALSE) {
cout << "Array pieno!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (search (c, val) == TRUE) {
cout << "Valore presente: " << val << endl;
      } else {
cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (empty (c) == TRUE) {
cout << "Array vuoto!" << endl;
      } else {
print (c);
      }
      break;
    case 'f':
      break;
    default:
      cout << "Opzione errata\n";
    }
  } while (res != 'f');

  return 0;
}
```

3 collezione_A32.h

```
// -*-C++ - *-
#define MAX_ARRAY_LENGTH 100

struct elemento {
  long val;
};

struct array {
  int dim;
  elemento *elementi[MAX_ARRAY_LENGTH];
};

typedef array *collezione;

enum boolean {
  FALSE, TRUE
};

void init (collezione &c);
boolean empty (const collezione &c);
boolean full (const collezione &c);
boolean insert (collezione &c, long val);
boolean search (const collezione &c, long val);
void print (const collezione &c);
```

3 soluzione_A32.cc

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

using namespace std;

void init (collezione &c) {
  // Inizializza la collezione
  c = new array;
  c->dim = 0;
}

boolean empty (const collezione &c) {
  // Ritorna "TRUE" se l'array  vuoto, "FALSE" altrimenti
  return (c->dim == 0) ? TRUE : FALSE;
}

boolean full (const collezione &c) {
  // Ritorna "TRUE" se l'array  pieno, "FALSE" altrimenti
  return (c->dim == MAX_ARRAY_LENGTH) ? TRUE : FALSE;
}

boolean insert (collezione &c, long val) {
  // Inserimento ordinato di un elemento (si assume l'array gi ordinato
  // in senso DECRESCENTE)
  if (full (c) == FALSE) {
    int indice = 0;
    if (empty (c) == FALSE) {
```

```
      while (indice < c->dim && val < c->elementi[indice]->val) {
        indice++;
      }
      // Sposta il contenuto dell'array a destra di una posizione
      for (int i = c->dim; i > indice; i--) {
        c->elementi[i] = c->elementi[i - 1];
      }
    }
    // Aggiunge una posizione all'array
    c->dim++;
    // Inserisce il contenuto nell'array
    elemento *e = new elemento;
    e->val = val;
    // L'elemento va inserito in posizione "i"
    c->elementi[indice] = e;
    return TRUE;
  }
  return FALSE;
}

boolean search_rec (const collezione &c, long val, int start, int end) {
  if (start > end) {
    return FALSE;
  }
  // Trovo il "centro" dell'array
  int center = start + (end - start) / 2;
  if (val == c->elementi[center]->val) {
    return TRUE;
  } else if (val > c->elementi[center]->val) {
    // Prendo il lato sinistro dell'array
    return search_rec (c, val, start, center - 1);
  } else {
    // Prendo il lato destro dell'array
    return search_rec (c, val, center + 1, end);
  }
}

boolean search (const collezione &c, long val) {
  if (empty (c) == TRUE) {
    return FALSE;
  } else {
    return search_rec (c, val, 0, c->dim - 1);
  }
}

void print (const collezione &c) {
  if (empty (c) == FALSE) {
    // Stampo tutti gli elementi dell'array
    cout << "Array: [";
    for (int i = 0; i < c->dim; i++) {
      cout << c->elementi[i]->val << " ";
    }
    cout << "]" << endl;
  } else {
```

```
    cout << "Array vuoto." << endl;
  }
}
```

3 Nel file `collezione_main.cc` è definita la funzione `main` che contiene un menù per gestire un array ordinato di puntatori a una **struct** chiamata "elemento". L'array ha dimensione massima prefissata. Ogni "elemento" contiene un valore `char` chiamato "val". Scrivere, in un nuovo file `collezione.cc`, le definizioni delle funzioni dichiarate nello header file `collezione.h` in modo tale che:

- `init` inizializzi la collezione;
- `empty` controlli se la collezione è vuota, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `full` controlli se la collezione è piena, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `insert` inserisca l'elemento passato come parametro nella collezione, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti. La collezione deve essere ordinata in maniera **crescente** e se l'elemento è già presente deve essere inserito ugualmente. Esempio: l'inserimento dei seguenti valori:

  ```
  a  z  n  l  q  q
  ```

  deve produrre il seguente array:

  ```
  a  l  n  q  q  z
  ```

- `search` cerchi nella collezione l'elemento passato in input, restituendo `TRUE` se l'elemento è presente, e `FALSE` altrimenti: la funzione deve richiedere un numero di passi al più logaritmico rispetto al numero di elementi effettivamente contenuti nell'array;
- `print` stampi a video il contenuto della collezione, in ordine **crescente**.

NOTA: non è ammesso alterare in alcun modo i file `collezione_main.cc` e `collezione.h`.

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

3 collezione_A33_main.cc

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

int  main () {
  char res;
  collezione c;
  char val;
  init (c);
  do {
    cout << "\nOperazioni possibili:\n"
      << " Inserimento (i)\n"
      << " Ricerca (r)\n"
      << " Stampa ordinata (s)\n"
      << " Fine (f)\n";
    cout << "Operazione selezionata: ";
    cin >> res;
    switch (res) {
    case 'i':
      cout << "Valore : ";
      cin >> val;
      if (insert (c, val) == FALSE) {
cout << "Array pieno!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (search (c, val) == TRUE) {
cout << "Valore presente: " << val << endl;
      } else {
cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (empty (c) == TRUE) {
cout << "Array vuoto!" << endl;
      } else {
print (c);
      }
      break;
    case 'f':
      break;
    default:
      cout << "Opzione errata\n";
    }
  } while (res != 'f');

  return 0;
}
```

3 collezione_A33.h

```
// -*-C++ - *-
#define MAX_ARRAY_LENGTH 100

struct elemento {
  char val;
};

struct array {
  int dim;
  elemento *elementi[MAX_ARRAY_LENGTH];
};

typedef array *collezione;

enum boolean {
  FALSE, TRUE
};

void init (collezione &c);
boolean empty (const collezione &c);
boolean full (const collezione &c);
boolean insert (collezione &c, char val);
boolean search (const collezione &c, char val);
void print (const collezione &c);
```

3 soluzione_A33.cc

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

using namespace std;

void init (collezione &c) {
  // Inizializza la collezione
  c = new array;
  c->dim = 0;
}

boolean empty (const collezione &c) {
  // Ritorna "TRUE" se l'array  vuoto, "FALSE" altrimenti
  return (c->dim == 0) ? TRUE : FALSE;
}

boolean full (const collezione &c) {
  // Ritorna "TRUE" se l'array  pieno, "FALSE" altrimenti
  return (c->dim == MAX_ARRAY_LENGTH) ? TRUE : FALSE;
}

boolean insert (collezione &c, char val) {
  // Inserimento ordinato di un elemento (si assume l'array gi ordinato
  // in senso CRESCENTE)
  if (full (c) == FALSE) {
    int indice = 0;
    if (empty (c) == FALSE) {
```

```
      while (indice < c->dim && val > c->elementi[indice]->val) {
        indice++;
      }
      // Sposta il contenuto dell'array a destra di una posizione
      for (int i = c->dim; i > indice; i--) {
        c->elementi[i] = c->elementi[i - 1];
      }
    }
    // Aggiunge una posizione all'array
    c->dim++;
    // Inserisce il contenuto nell'array
    elemento *e = new elemento;
    e->val = val;
    // L'elemento va inserito in posizione "i"
    c->elementi[indice] = e;
    return TRUE;
  }
  return FALSE;
}

boolean search_rec (const collezione &c, char val, int start, int end) {
  if (start > end) {
    return FALSE;
  }
  // Trovo il "centro" dell'array
  int center = start + (end - start) / 2;
  if (val == c->elementi[center]->val) {
    return TRUE;
  } else if (val < c->elementi[center]->val) {
    // Prendo il lato sinistro dell'array
    return search_rec (c, val, start, center - 1);
  } else {
    // Prendo il lato destro dell'array
    return search_rec (c, val, center + 1, end);
  }
}

boolean search (const collezione &c, char val) {
  if (empty (c) == TRUE) {
    return FALSE;
  } else {
    return search_rec (c, val, 0, c->dim - 1);
  }
}

void print (const collezione &c) {
  if (empty (c) == FALSE) {
    // Stampo tutti gli elementi dell'array
    cout << "Array: [";
    for (int i = 0; i < c->dim; i++) {
      cout << c->elementi[i]->val << " ";
    }
    cout << "]" << endl;
  } else {
```

```
    cout << "Array vuoto." << endl;
  }
}
```

3 Nel file `collezione_main.cc` è definita la funzione `main` che contiene un menù per gestire un array ordinato di puntatori a una **struct** chiamata "elemento". L'array ha dimensione massima prefissata. Ogni "elemento" contiene un valore `double` chiamato "val". Scrivere, in un nuovo file `collezione.cc`, le definizioni delle funzioni dichiarate nello header file `collezione.h` in modo tale che:

- `init` inizializzi la collezione;
- `empty` controlli se la collezione è vuota, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `full` controlli se la collezione è piena, restituendo `TRUE` in caso affermativo e `FALSE` in caso contrario;
- `insert` inserisca l'elemento passato come parametro nella collezione, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti. La collezione deve essere ordinata in maniera **decrescente** e se l'elemento è già presente deve essere inserito ugualmente. Esempio: l'inserimento dei seguenti valori:

  ```
  3.14  9.97  1  12  1000  9.97
  ```

  deve produrre il seguente array:

  ```
  1000  12  9.97  9.97  3.14  1
  ```

- `search` cerchi nella collezione l'elemento passato in input, restituendo `TRUE` se l'elemento è presente, e `FALSE` altrimenti: la funzione deve richiedere un numero di passi al più logaritmico rispetto al numero di elementi effettivamente contenuti nell'array;
- `print` stampi a video il contenuto della collezione, in ordine **decrescente**.

NOTA: non è ammesso alterare in alcun modo i file `collezione_main.cc` e `collezione.h`.

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

3 collezione_A34_main.cc

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

int  main () {
  char res;
  collezione c;
  double val;
  init (c);
  do {
    cout << "\nOperazioni possibili:\n"
      << " Inserimento (i)\n"
      << " Ricerca (r)\n"
      << " Stampa ordinata (s)\n"
      << " Fine (f)\n";
    cout << "Operazione selezionata: ";
    cin >> res;
    switch (res) {
    case 'i':
      cout << "Valore : ";
      cin >> val;
      if (insert (c, val) == FALSE) {
cout << "Array pieno!" << endl;
      }
      break;
    case 'r':
      cout << "Valore: ";
      cin >> val;
      if (search (c, val) == TRUE) {
cout << "Valore presente: " << val << endl;
      } else {
cout << "Valore non presente" << endl;
      }
      break;
    case 's':
      if (empty (c) == TRUE) {
cout << "Array vuoto!" << endl;
      } else {
print (c);
      }
      break;
    case 'f':
      break;
    default:
      cout << "Opzione errata\n";
    }
  } while (res != 'f');

  return 0;
}
```

3 collezione_A34.h

```
// -*-C++ - *-
#define MAX_ARRAY_LENGTH 100

struct elemento {
  double val;
};

struct array {
  int dim;
  elemento *elementi[MAX_ARRAY_LENGTH];
};

typedef array *collezione;

enum boolean {
  FALSE, TRUE
};

void init (collezione &c);
boolean empty (const collezione &c);
boolean full (const collezione &c);
boolean insert (collezione &c, double val);
boolean search (const collezione &c, double val);
void print (const collezione &c);
```

3 soluzione_A34.cc

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

using namespace std;

void init (collezione &c) {
  // Inizializza la collezione
  c = new array;
  c->dim = 0;
}

boolean empty (const collezione &c) {
  // Ritorna "TRUE" se l'array  vuoto, "FALSE" altrimenti
  return (c->dim == 0) ? TRUE : FALSE;
}

boolean full (const collezione &c) {
  // Ritorna "TRUE" se l'array  pieno, "FALSE" altrimenti
  return (c->dim == MAX_ARRAY_LENGTH) ? TRUE : FALSE;
}

boolean insert (collezione &c, double val) {
  // Inserimento ordinato di un elemento (si assume l'array gi ordinato
  // in senso DECRESCENTE)
  if (full (c) == FALSE) {
    int indice = 0;
    if (empty (c) == FALSE) {
```

```
      while (indice < c->dim && val < c->elementi[indice]->val) {
        indice++;
      }
      // Sposta il contenuto dell'array a destra di una posizione
      for (int i = c->dim; i > indice; i--) {
        c->elementi[i] = c->elementi[i - 1];
      }
    }
    // Aggiunge una posizione all'array
    c->dim++;
    // Inserisce il contenuto nell'array
    elemento *e = new elemento;
    e->val = val;
    // L'elemento va inserito in posizione "i"
    c->elementi[indice] = e;
    return TRUE;
  }
  return FALSE;
}

boolean search_rec (const collezione &c, double val, int start, int end) {
  if (start > end) {
    return FALSE;
  }
  // Trovo il "centro" dell'array
  int center = start + (end - start) / 2;
  if (val == c->elementi[center]->val) {
    return TRUE;
  } else if (val > c->elementi[center]->val) {
    // Prendo il lato sinistro dell'array
    return search_rec (c, val, start, center - 1);
  } else {
    // Prendo il lato destro dell'array
    return search_rec (c, val, center + 1, end);
  }
}

boolean search (const collezione &c, double val) {
  if (empty (c) == TRUE) {
    return FALSE;
  } else {
    return search_rec (c, val, 0, c->dim - 1);
  }
}

void print (const collezione &c) {
  if (empty (c) == FALSE) {
    // Stampo tutti gli elementi dell'array
    cout << "Array: [";
    for (int i = 0; i < c->dim; i++) {
      cout << c->elementi[i]->val << " ";
    }
    cout << "]" << endl;
  } else {
```

```
    cout << "Array vuoto." << endl;
  }
}
```