Quarto Appello di Programmazione I

08 Luglio 2010
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` la dichiarazione e la definizione della funzione **ricorsiva** `array_palindromo` che verifica in modo **ricorsivo** se una sequenza di **numeri** contenuta in un array di al più 100 elementi di tipo `int` è palindroma. Il contenuto di tale array è considerato palindromo se il primo elemento è uguale all'ultimo, il secondo elemento è uguale al penultimo, e così via... per tutti gli elementi dell'array.

Ad esempio il seguente array:

```
9 22 5 5 22 9
```

è palindromo in quanto il primo elemento (`9`) è uguale all'ultimo, il secondo elemento (`22`) è uguale al penultimo, ed infine il terzo elemento (`5`) è uguale al terzultimo. Anche il seguente array di 5 elementi è palindromo:

```
0 8 33 8 0
```

Infatti, in caso di numero dispari di elementi, l'elemento centrale è ininfluente.
Invece, l'array:

```
10 2 7 19 4 0 4 18 7 2 10
```

non è palindromo, infatti il quarto numero a partire dalla testa dell'array (`19`) è diverso dal quarto numero a partire dal fondo (ovvero `18`).

NOTA: **La funzione deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli; può invece fare uso di eventuali funzioni ausiliarie purchè a loro volta ricorsive.**

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

1 **esercizio1.cc**

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

// Inserire qui le dichiarazioni

bool array_palindromo (int *sequenza, int dim);

int main()
{
  int dim;
  int array[100];

  do
  {
    cout << "Inserisci il numero (0 < ... < 100) di elementi da memorizzare nell'array: ";
    cin >> dim;
  } while (dim <= 0);

  if (dim > 100)
  {
    cout << "Puoi inserire solo 100 elementi!" << endl;
    dim = 100;
  }
  for (int i = 0; i < dim; i++)
  {
    cout << "Inserire intero " << i+1 <<" : ";
    cin >> array[i];
  }

  if ( array_palindromo(array, dim) )
    cout << "La sequenza inserita e' palindroma!" << endl;
  else
    cout << "La sequenza inserita NON e' palindroma!" << endl;

  return 0;
}

// Inserire qui le definizioni

bool array_palindromo (int *sequenza, int dim)
{
  if (dim < 1)
    return true;
  else if (sequenza[0] != sequenza[dim-1])
    return false;
  else
    return array_palindromo(sequenza+1, dim-2);
}

// SOLUZIONE ALTERNATIVA
/*
bool array_palindromo (int *sequenza, int primo, int ultimo)
```

```
{
  if (primo > ultimo)
    return true;
  else if (sequenza[primo] != sequenza[ultimo])
    return false;
  else
    return array_palindromo(sequenza, primo+1, ultimo-1);
}

bool array_palindromo (int *sequenza, int dim)
{
  return array_palindromo(sequenza, 0, dim-1);
}
*/
```

1 Scrivere nel file `esercizio1.cc` la dichiarazione e la definizione della funzione **ricorsiva** `sequenza_palindroma` che verifica in modo **ricorsivo** se una sequenza di **caratteri** contenuta in un array di al più 100 elementi di tipo `char` è palindroma. Il contenuto di tale array è considerato palindromo se il primo elemento è uguale all'ultimo, il secondo elemento è uguale al penultimo, e così via... per tutti gli elementi dell'array.

Ad esempio il seguente array:

```
a # 9 g F F G 9 # a
```

non è palindromo, infatti il quarto carattere a partire dalla testa dell'array (`g`) è diverso dal quarto carattere a partire dal fondo (ovvero `G`).
L'array di 5 elementi:

```
X 1 $ 1 X
```

contiene, invece, una sequenza di caratteri palindroma (si noti che, in caso di numero dispari di elementi, l'elemento centrale è ininfluente) così come è palindromo anche il contenuto dell'array:

```
m @ 4 4 @ m
```

visto che il primo elemento (`m`) è uguale all'ultimo, il secondo elemento (`@`) è uguale al penultimo, ed infine il terzo elemento (`4`) è uguale al terzultimo.

NOTA: **La funzione deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli; può invece fare uso di eventuali funzioni ausiliarie purchè a loro volta ricorsive.**

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

1 **esercizio1.cc**

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

// Inserire qui le dichiarazioni

bool sequenza_palindroma (char *sequenza, int dim);

int main()
{
  int dim;
  char array[100];

  do
  {
    cout << "Inserisci il numero (0 < ... < 100) di elementi da memorizzare nell'array: ";
    cin >> dim;
  } while (dim <= 0);

  if (dim > 100)
  {
    cout << "Puoi inserire solo 100 elementi!" << endl;
    dim = 100;
  }
  for (int i = 0; i < dim; i++)
  {
    cout << "Inserire carattere " << i+1 <<" : ";
    cin >> array[i];
  }

  if ( sequenza_palindroma(array, dim) )
    cout << "La sequenza inserita e' palindroma!" << endl;
  else
    cout << "La sequenza inserita NON e' palindroma!" << endl;

  return 0;
}

// Inserire qui le definizioni

bool sequenza_palindroma (char *sequenza, int dim)
{
  if (dim < 1)
    return true;
  else if (sequenza[0] != sequenza[dim-1])
    return false;
  else
    return sequenza_palindroma(sequenza+1, dim-2);
}

// SOLUZIONE ALTERNATIVA
/*
bool sequenza_palindroma (char *sequenza, int primo, int ultimo)
```

```
{
  if (primo > ultimo)
    return true;
  else if (sequenza[primo] != sequenza[ultimo])
    return false;
  else
    return sequenza_palindroma(sequenza, primo+1, ultimo-1);
}

bool sequenza_palindroma (char *sequenza, int dim)
{
  return sequenza_palindroma(sequenza, 0, dim-1);
}
*/
```

2 Scrivere nel file `esercizio2.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.

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.

Dato, 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 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 **esercizio2.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);
}
```

2 Scrivere nel file `esercizio2.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.

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.

Dato, 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 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 **esercizio2.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);
}
```

3 Nel file `lista_main.cc` è definita la funzione `main` che contiene un menu per gestire una **lista concatenata semplice** di **interi**. La lista può contenere un numero indefinito di elementi.

Scrivere, in un nuovo file `lista.cc`, le definizioni delle funzioni dichiarate nello header file `lista.h` in modo tale che:

- `init` inizializzi la lista;
- `deinit` liberi la memoria utilizzata dalla lista;
- `ins_testa` inserisca l'elemento passato come parametro in testa alla lista;
- `estr_testa` tolga l'elemento in testa alla lista e lo memorizzi nella variabile passata come parametro;
- `cerca_elem` cerchi nella lista l'elemento passato come parametro e ritorni `OK` se tale emento è presente e `FAIL` altrimenti;
- `stampa_lista` stampi a video il contenuto della lista.

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

3 lista_main.cc

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


int main()
{
    char res;
    int num;
    node *q;

  init(q);
    do {
        cout << "\nOperazioni possibili:\n"
             << "Inserimento in testa (i)\n"
             << "Estrazione in testa (e)\n"
             << "Ricerca elemento (r)\n"
 << "Stampa (s)\n"
             << "Fine (f)\n";
        cin >> res;
        switch (res) {
        case 'i':
            cout << "Valore: ";
            cin >> num;
ins_testa(q,num);
            break;
        case 'e':
            if (estr_testa(q, num) == FAIL) {
                cout << "Lista vuota\n";
            } else {
                cout << "Val: " << num << endl;
            }
break;
case 'r':
            cout << "Valore: ";
            cin >> num;
            if (cerca_elem(q, num) == FAIL) {
                cout << "Non Trovato!" << endl;
            } else {
                cout << "Trovato!" << endl;
            }
break;
        case 's':
            stampa_lista(q);
            break;
        case 'f':
            break;
        default:
            cout << "Valore errato!\n";
        }
    } while (res != 'f');
```

```
deinit(q);

    return 0;
}
```

3 lista.h

```
#ifndef STRUCT_LISTA_H
#define STRUCT_LISTA_H

// dichiarazioni per la gestine della lista di int

enum retval { FAIL, OK };

struct node {
int item;
node* next;
};

void init(node *&punt);
void deinit(node *&punt);
void ins_testa(node *&punt, int a);
retval estr_testa(node *&punt, int& a);
retval cerca_elem(node *&punt, int a);
void stampa_lista(node *&punt);


#endif
```

3 soluzione_A31.cc

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

using namespace std;


void init(node *&punt) {
punt = NULL;
}

void ins_testa(node *&punt, int a) {
node *p = new node;
p->item = a;
p->next = punt;
punt = p;
}

retval estr_testa(node *&punt, int& a) {
node *p = punt;
if (punt!=NULL) {
```

```
a = punt->item ;
punt = punt->next;
delete p;
}
if (punt!=NULL){
return OK;
} else {
return FAIL;
}
}

retval cerca_elem(node *&punt, int a) {
node *r;
for (r = punt; r!=NULL && r->item != a; r = r->next);
if (r!=NULL) {
return OK;
} else {
return FAIL;
}
}

void stampa_lista(node *&punt) {
node *r;
cout << "Lista:";
for (r = punt; r!=NULL; r = r->next){
cout << " " << r->item;
}
cout << "\n";
}

void deinit(node *&punt){
while(punt!=NULL){
node* p = punt;
punt = punt->next;
delete p;
}
}
```

3 Nel file `lista_main.cc` è definita la funzione `main` che contiene un menu per gestire una **lista concatenata semplice** di **float**. La lista può contenere un numero indefinito di elementi.

Scrivere, in un nuovo file `lista.cc`, le definizioni delle funzioni dichiarate nello header file `lista.h` in modo tale che:

- `init` inizializzi la lista;
- `deinit` liberi la memoria utilizzata dalla lista;
- `ins_testa` inserisca l'elemento passato come parametro in testa alla lista;
- `estr_testa` tolga l'elemento in testa alla lista e lo memorizzi nella variabile passata come parametro;
- `cerca_elem` cerchi nella lista l'elemento passato come parametro e ritorni `OK` se tale emento è presente e `FAIL` altrimenti;
- `stampa_lista` stampi a video il contenuto della lista.

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

3 lista_main.cc

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


int main()
{
    char res;
    float num;
    node *q;

init(q);
    do {
        cout << "\nOperazioni possibili:\n"
             << "Inserimento in testa (i)\n"
             << "Estrazione in testa (e)\n"
             << "Ricerca elemento (r)\n"
 << "Stampa (s)\n"
             << "Fine (f)\n";
        cin >> res;
        switch (res) {
        case 'i':
            cout << "Valore: ";
            cin >> num;
ins_testa(q,num);
            break;
        case 'e':
            if (estr_testa(q, num) == FAIL) {
                cout << "Lista vuota\n";
            } else {
                cout << "Val: " << num << endl;
            }
break;
case 'r':
            cout << "Valore: ";
            cin >> num;
            if (cerca_elem(q, num) == FAIL) {
                cout << "Non Trovato!" << endl;
            } else {
                cout << "Trovato!" << endl;
            }
break;
        case 's':
            stampa_lista(q);
            break;
        case 'f':
            break;
        default:
            cout << "Valore errato!\n";
        }
    } while (res != 'f');
```

```
deinit(q);

    return 0;
}
```

3 lista.h

```
#ifndef STRUCT_LISTA_H
#define STRUCT_LISTA_H

// dichiarazioni per la gestine della lista di float

enum retval { FAIL, OK };

struct node {
float item;
node* next;
};

void init(node *&punt);
void deinit(node *&punt);
void ins_testa(node *&punt, float a);
retval estr_testa(node *&punt, float& a);
retval cerca_elem(node *&punt, float a);
void stampa_lista(node *&punt);

#endif
```

3 soluzione_A32.cc

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

using namespace std;


void init(node *&punt) {
punt = NULL;
}

void ins_testa(node *&punt, float a) {
node *p = new node;
p->item = a;
p->next = punt;
punt = p;
}

retval estr_testa(node *&punt, float& a) {
node *p = punt;
if (punt!=NULL) {
a = punt->item ;
```

```
punt = punt->next;
delete p;
}
if (punt!=NULL){
return OK;
} else {
return FAIL;
}
}

retval cerca_elem(node *&punt, float a) {
node *r;
for (r = punt; r!=NULL && r->item != a; r = r->next);
if (r!=NULL) {
return OK;
} else {
return FAIL;
}
}

void stampa_lista(node *&punt) {
node *r;
cout << "Lista:";
for (r = punt; r!=NULL; r = r->next){
cout << " " << r->item;
}
cout << "\n";
}

void deinit(node *&punt){
while(punt!=NULL){
node* p = punt;
punt = punt->next;
delete p;
}
}
```