Quarto Appello di Programmazione I

04 Settembre 2007
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 Il moto in caduta libera di un oggetto nel vuoto è descritto tramite la seguente formula:

$$spazio\,percorso = 0.5 * 9.8 * tempo^2$$

Per calcolare quanto tempo un oggetto impiega a cadere da una certa altezza, è necessario utilizzare la seguente formula:

$$tempo = \sqrt{\frac{2 * altezza}{9.8}}$$

.

Nel file `esercizio1.cc` sono definite le seguenti funzioni:

- la funzione `stampa_altezze` che presi in input un array di numeri reali ed un intero, stampa a video il contenuto dell'array specificato;
- la funzione `main` che dichiarte due variabili di tipo `int` ed un puntatore a numeri reali, richiede in input da tastiera un numero intero, invoca la funzione `calcola_altezze` ed infine chiama la funzione `stampa_altezze`.

Scrivere la funzione `calcola_altezze` in modo tale che, preso in input la variabile intera `altezza_iniziale` corrispondente all'altezza iniziale di un oggetto, restituisca un array contenente, per ogni secondo compreso l'istante zero, il valore dell'altezza a cui si trova l'oggetto in caduta libera nel vuoto fino a quando arriva a terra. In particolare nella cella `0` ci sarà il valore dell'altezza iniziale, nella cella `1` ci sarà l'altezza dell'oggetto dopo un secondo e così di seguito.

Ad esempio, se il valore di `altezza_iniziale` è pari a `100`, il contenuto del nuovo array generato dalla funzione `calcola_altezze` sarà:
`100 95.1 80.4 55.9 21.6`

NOTA: all'interno di questo programma non è ammesso l'utilizzo di variabili globali o di tipo `static`.
Per semplicità supporre che l'array `altezze`:

- non sia mai vuoto,
- abbia sempre un numero di interi inferiore a 100.

VALUTAZIONE: questo esercizio vale 4 punti (al punteggio di tutti gli esercizi va poi sommato 12).

1 **esercizio1.cc**

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

void stampa_altezze (double altezze[100], int dimensione);
double* calcola_altezze (int altezza_iniziale, int& dimensione);

int main(){

  int dimensione, altezza_iniziale;;
  double *altezze;

  cout << "Inserisci l'altezza inziale dell'oggetto (in metri): ";
  cin >> altezza_iniziale;

  altezze = calcola_altezze(altezza_iniziale, dimensione);

  stampa_altezze(altezze, dimensione);
  return(0);
}

void stampa_altezze (double altezze[100], int dimensione)
{
  int i;

  for(i=0; i<=dimensione; i++)
    {
      cout << "Dopo " << i << " secondi, l'oggetto e' ad un altezza pari a " << altezze[i] <
    }
  cout<<endl;
}

//Scrivete il vostro codice al di sotto di questo commento

double* calcola_altezze (int altezza_iniziale, int& dimensione)
{
  int i,tempo;
  double *altezze;

  tempo = (int) sqrt((2*altezza_iniziale)/9.8);
  dimensione=tempo;

  altezze = new double[tempo+1];

  for(i=0;i<=tempo;i++){
    altezze[i]=altezza_iniziale-(0.5*9.8*i*i);
  }
  return altezze;
}
```

1 Il moto in caduta libera di un oggetto nel vuoto è descritto tramite la seguente formula:

$$spazio\,percorso = 0.5 * 9.8 * tempo^2$$

Per calcolare quanto tempo un oggetto impiega a cadere da una certa altezza, è necessario utilizzare la seguente formula:

$$tempo = \sqrt{\frac{2 * altezza}{9.8}}$$

.

Nel file `esercizio1.cc` sono definite le seguenti funzioni:

- la funzione `stampa_spazio_percorso` che presi in input un array di numeri reali ed un intero, stampa a video il contenuto dell'array specificato;
- la funzione `main` che dichiarate due variabili di tipo `int` ed un puntatore a numeri reali, richiede in input da tastiera un numero intero, invoca la funzione `calcola_spazio_percorso` ed infine chiama la funzione `stampa_spazio_percorso`.

Scrivere la funzione `calcola_spazio_percorso` in modo tale che, preso in input la variabile intera `altezza_iniziale` corrispondente all'altezza iniziale di un oggetto, restituisca un array contenente, per ogni secondo compreso l'istante zero, il valore dello spazio percorso dall'oggetto in caduta libera nel vuoto fino a quando arriva a terra. In particolare nella cella `0` ci sarà il valore `0`, nella cella `1` ci sarà lo spazio percorso dell'oggetto dopo un secondo e così di seguito.

Ad esempio, se il valore di `altezza_iniziale` è pari a `100`, il contenuto del nuovo array generato dalla funzione `calcola_spazio_percorso` sarà:
`0 4.9 19.6 44.1 78.4`

NOTA: all'interno di questo programma non è ammesso l'utilizzo di variabili globali o di tipo `static`.
Per semplicità supporre che l'array `spazio_percorso`:

- non sia mai vuoto,
- abbia sempre un numero di interi inferiore a 100.

VALUTAZIONE: questo esercizio vale 4 punti (al punteggio di tutti gli esercizi va poi sommato 12).

1 **esercizio1.cc**

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

void stampa_spazio_percorso (double spazio_percorso[100], int dimensione);
double* calcola_spazio_percorso (int altezza_iniziale, int& dimensione);

int main(){

  int dimensione, altezza_iniziale;;
  double *spazio_percorso;

  cout << "Inserisci l'altezza inziale dell'oggetto (in metri): ";
  cin >> altezza_iniziale;

  spazio_percorso = calcola_spazio_percorso(altezza_iniziale, dimensione);

  stampa_spazio_percorso(spazio_percorso, dimensione);
  return(0);
}

void stampa_spazio_percorso (double spazio_percorso[100], int dimensione)
{
  int i;

  for(i=0; i<=dimensione; i++)
    {
      cout << "Dopo " << i << " secondi, l'oggetto ha percorso uno spazio pari a " << spazio
    }
  cout<<endl;
}

//Scrivete il vostro codice al di sotto di questo commento

double* calcola_spazio_percorso (int altezza_iniziale, int& dimensione)
{
  int i,tempo;
  double *spazio_percorso;

  tempo = (int) sqrt((2*altezza_iniziale)/9.8);
  dimensione=tempo;

  spazio_percorso = new double[tempo+1];

  for(i=0;i<=tempo;i++){
    spazio_percorso[i]=(0.5*9.8*i*i);
  }
  return spazio_percorso;
}
```

2 Nel file `esercizio2.cc` scrivere un programma che, presa come argomento del `main` una lista di nomi di file, copi alla fine del primo file della lista il contenuto dei successivi file in ordine.

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

```
./a.out testo1 testo2 testo3 testo4 testo5
```

aggiungerà alla fine del file `testo1`, il contenuto dei file `testo2`, `testo3`, `testo4` e `testo5`, esattamente in quest'ordine.

Per semplicità si assuma che il testo contenuto nei file sia su un'unica riga.

VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 12).

2 **esercizio2.cc**

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

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

  fstream * my_file;
  char c;

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

  my_file = new fstream [argc-1];

  my_file[0].open(argv[1],ios::out|ios::app);
  for (int i=1; i<argc-1; i++) {
    my_file[i].open(argv[i+1],ios::in);

    while (my_file[i].get(c)) {
      my_file[0].put(c);
    }

    my_file[i].close();
  }
  my_file[0].close();

  return(0);
}
```

2 Nel file `esercizio2.cc` scrivere un programma che, presa come argomento del `main` una lista di nomi di file, copi alla fine di ogni file il contenuto del file che lo succede nella lista (se tale file esiste).

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

```
./a.out testo1 testo2 testo3 testo4 testo5
```

aggiungerà alla fine del file `testo1` il contenuto dei file `testo2`, alla fine del file `testo2` il contenuto dei file `testo3`, alla fine del file `testo3` il contenuto dei file `testo4`, ed alla fine del file `testo4` il contenuto dei file `testo5`.

Per semplicità si assuma che il testo contenuto nei file sia su un'unica riga.

VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 12).

2 **esercizio2.cc**

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

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

  fstream * my_file;
  char c;

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

  my_file = new fstream [argc-1];

  for (int i=0; i<argc-2; i++) {
    my_file[i].open(argv[i+1],ios::out|ios::app);
    my_file[i+1].open(argv[i+2],ios::in);

    while (my_file[i+1].get(c)) {
      my_file[i].put(c);
    }

    my_file[i].close();
    my_file[i+1].close();
  }

  return(0);
}
```

3 Nel file `queue_main.cc` è definita la funzione `main` che contiene un menu per gestire una coda di puntatori a elementi di tipo struct `studente` tramite array. La struct `studente` e le funzioni per gestirla sono dichiarate nel file `studente.h` e definite nel file binario `studente.o`. Scrivere, in un nuovo file `queue.cc`, le definizioni delle funzioni dichiarate nello header file `queue.h` in modo tale che:

- `init` inizializzi la coda;
- `enqueue` inserisca l'elemento passato come parametro nella coda;
- `dequeue` tolga l'elemento in testa alla coda e lo memorizzi nella variabile passata come parametro;
- `stampa` stampi a video il contenuto della coda.

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

3 queue_main.cc

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


int main ()
{
  char res;
  char n[20], c[20];
  studente *p;
  queue q;
  int matricola;

  init(q);
  do {
    cout << "\nOperazioni possibili:\n"
         << "Enqueue (e)\n"
         << "Dequeue (d)\n"
         << "Print (p)\n"
         << "Fine (f)\n";
    cin >> res;
    switch (res) {
    case 'e':
      cout << "Nome: ";
      cin >> n;
      cout << "Cognome: ";
      cin >> c;
      cout << "Matricola: ";
      cin >> matricola;
      p = crea_studente(n,c,matricola);
      if (p == NULL)
cout << "Memoria piena\n";
      if (enqueue(p,q)==FAIL) {
        cout << "Coda piena\n";
elimina_studente(p);
      }
      break;
    case 'd':
      if (dequeue(p,q)==FAIL)
        cout << "Coda vuota\n";
      else {
        stampa_studente(p);
elimina_studente(p);
      }
      break;
    case 'p':
      print(q);
      break;
    case 'f':
      break;
    default:
```

```
      cout << "Valore errato!\n";
    }
  } while (res != 'f');
}
```

3 **queue.h**

```
#ifndef STRUCT_QUEUE_H
#define STRUCT_QUEUE_H

// dichiarazioni per la gestine della coda di interi

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

static const int dim = 100;

struct queue
{
  int head, tail;
  studente * elem[dim];
};

enum retval { FAIL, OK };
enum boolean { FALSE, TRUE };

void init (queue &);
retval enqueue(studente *,queue &);
retval dequeue(studente * &,queue &);
void print (const queue &);

#endif
```

3 **queue.cc**

```
#include "queue.h"

// funzioni per la gestine di una coda di struct studente

static int next(int index)
{
  return (index+1)%dim;
}

void init(queue & q)
{
  q.tail=q.head=0;
}

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

```
static boolean fullp(const queue & q)
{
  return (boolean) (next(q.tail)==q.head);
}

retval enqueue (studente * p,queue & q)
{
  if (fullp(q))
    return FAIL;
  q.elem[q.tail] = p;
  q.tail = next(q.tail);
  return OK;
}

retval dequeue(studente * & p,queue & q)
{
  if (emptyp(q))
    return FAIL;
  p = q.elem[q.head];
  q.head = next(q.head);
  return OK;
}

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

3 studente.h

```
#ifndef STUDENTE_H
#define STUDENTE_H

#include <iostream>
struct studente;

void stampa_studente(studente *p);
studente* crea_studente(char * n, char * c, int matricola);
void elimina_studente(studente *p);

#endif
```

3 studente.cc

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

struct studente {
  char nome[20];
```

```
  char cognome[20];
  int matricola;
};

void stampa_studente(studente *p) {
  //  cout << "Nome: " << p->nome << " ";
  //  cout << "Cognome: " << p->cognome << endl;
  cout << p->nome << " " << p->cognome << " " << p->matricola << endl;
}

studente* crea_studente(char * n, char * c, int m) {
  studente *np = new studente;
  if (np!=NULL){
    strcpy(np->nome, n);
    strcpy(np->cognome, c);
    np->matricola=m;
  }
  return np;
}

void elimina_studente(studente *p) {
  delete p;
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menu per gestire una pila di puntatori a elementi di tipo struct `studente` tramite array. La struct `studente` e le funzioni per gestirla sono dichiarate nel file `studente.h` e definite nel file binario `studente.o`. 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;
- `push` inserisca l'elemento passato come parametro nella pila;
- `pop` tolga l'elemento in testa alla pila e lo memorizzi nella variabile passata come parametro;
- `top` legga l'elemento in testa alla pila e lo memorizzi nella variabile passata come parametro;
- `stampa` stampi a video il contenuto della pila.

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

3 stack_main.cc

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


int main ()
{
  char res;
  char n[20], c[20];
  studente *p;
  stack s;
  int matricola;

  init(s);
  do {
    cout << "\nOperazioni possibili:\n"
         << " Push (u)\n"
         << " Pop (o)\n"
         << " Top (t)\n"
         << " Print (p)\n"
 << " Fine (f)\n";
    cin >> res;
    switch (res) {
    case 'u':
      cout << "Nome: ";
      cin >> n;
      cout << "Cognome: ";
      cin >> c;
      cout << "Matricola: ";
      cin >> matricola;
      p = crea_studente(n,c,matricola);
      if (p == NULL)
cout << "Memoria piena\n";
      if (push(p,s)==FAIL) {
cout << "Memoria esaurita!\n";
elimina_studente(p);
      }
      break;
    case 'o':
      if (pop(p,s)==FAIL)
cout << "Stack vuoto!\n";
      else
elimina_studente(p);
      break;
    case 't':
      if (top(p,s)==FAIL)
cout << "Stack vuoto!\n";
      else
stampa_studente(p);
      break;
    case 'p':
```

```
      print(s);
      break;
    case 'f':
      break;
    default:
      cout << "Opzione errata\n";
    }
  } while (res != 'f');
}
```

3 **stack.h**

```
#ifndef STRUCT_STACK_H
#define STRUCT_STACK_H

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

enum retval {FAIL,OK};
enum boolean {FALSE,TRUE};

static const int dim = 100;

struct stack
{
  int indice;
  studente * elem[dim];
};

void init(stack & );
retval push (studente *, stack &);
retval top (studente * &, const stack &);
retval pop (studente * &, stack &);
void print(const stack &);

#endif
```

3 **stack.cc**

```
#include "stack.h"

// funzioni per la gestine di una pila di struct studente

static boolean emptyp (const stack & s)
{
  return (boolean) (s.indice==0);
}

static boolean fullp (const stack & s)
{
  return (boolean) (s.indice==dim);
}
```

```
void init (stack & s)
{
  s.indice = 0;
}

retval top (studente * &n, const stack & s)
{
  retval res;
  if (emptyp(s))
    res = FAIL;
  else {
    n=s.elem[s.indice-1];
    res = OK;
  }
  return res;
}

retval push (studente * n, stack & s)
{
  retval res;
  if (fullp(s))
    res = FAIL;
  else {
    s.elem[s.indice++]=n;
    res = OK;
  }
  return res;
}

retval pop (studente * &n, stack & s)
{
  retval res;
  if (emptyp(s))
    res = FAIL;
  else {
    n=s.elem[s.indice-1];
    s.indice--;
    res = OK;
  }
  return res;
}

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

3 studente.h

```
#ifndef STUDENTE_H
#define STUDENTE_H
```

```
#include <iostream>
struct studente;

void stampa_studente(studente *p);
studente* crea_studente(char * n, char * c, int matricola);
void elimina_studente(studente *p);

#endif
```

3 studente.cc

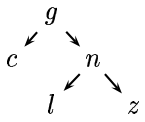
```
#include <iostream>
#include <cstring>
using namespace std;
#include "studente.h"

struct studente {
  char nome[20];
  char cognome[20];
  int matricola;
};

void stampa_studente(studente *p) {
  //  cout << "Nome: " << p->nome << " ";
  //  cout << "Cognome: " << p->cognome << endl;
  cout << p->nome << " " << p->cognome << " " << p->matricola << endl;
}

studente* crea_studente(char * n, char * c, int m) {
  studente *np = new studente;
  if (np!=NULL){
    strcpy(np->nome, n);
    strcpy(np->cognome, c);
    np->matricola=m;
  }
  return np;
}

void elimina_studente(studente *p) {
  delete p;
}
```

4 Dati i seguenti file:

- `tree.h` e `tree.o`, che implementano una libreria "albero di ricerca binaria di caratteri", a cui manca la funzione di stampa ordinata dell'albero;
- `stack.h` e `stack.o`, che implementano una libreria "stack di alberi";
- `esercizio4.cc` che contiene la funzione `main` con il menu che gestisce l'albero di ricerca binaria;

scrivere all'interno del file `esercizio4.cc` la definizione della funzione **iterativa** "`stampa`" che, preso in input un albero di ricerca binaria, ne stampi il contenuto in modo ordinato crescente.

Ad esempio, il seguente albero:



deve essere stampato nell'ordine 'c', 'g', 'l', 'n', 'z'.

N.B.: **La funzione non può essere ricorsiva: al suo interno, non ci possono quindi essere chiamate a sè stessa o ad altre funzioni ricorsive o mutualmente ricorsive.**

SUGGERIMENTO: utilizzare lo stack offerto nei file `stack.h` e `stack.o`.

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

4 esercizio4.cc

```
#include <iostream>
#include "tree.h"
#include "stack.h"

using namespace std;

retval stampa(tree t);

int main()
{
  char res,val;
  tree t, tmp;
  init(t);
  do {
    cout << "\nOperazioni possibili:\n"
         << "Inserimento (i)\n"
         << "Ricerca (r)\n"
         << "Stampa DFS (s)\n"
         << "Fine (f)\n";
    cin >> res;
    switch (res) {
    case 'i':
      cout << "Val? : ";
      cin >> val;
      insert(t,val);
      if (t==NULL)
        cout << "memoria esaurita!\n";
      break;
    case 'r':
      cout << "Val? : ";
      cin >> val;
      tmp=cerca(t,val);
      if (tmp!=NULL)
        cout << "Valore trovato!: " << val << endl;
      else
        cout << "Valore non trovato!\n";
      break;
    case 's':
      cout << "Stampa:\n";
      stampa(t);
      break;
    case 'f':
      break;
    default:
      cout << "Optione errata\n";
    }
  } while (res != 'f');
}

retval stampa(tree t) {
  stack_init();
  tree current = t;
```

```
  while(!vuoto(current) || !stack_vuoto()) {
    if (current!=NULL) {
      stack_push(current);
      current= current->left;
    }
    else { // !stack_empty()
      stack_top(current);
      stack_pop();
      tree_node_print(current);
      current=current->right;
    }
  }
}
```