Laboratorio Tecnologie dell' Informazione

PROVA SCRITTA – 16 Luglio 2009
Parte I – TEORIA

1 (10 punti)

Si descriva cosa sono i meccanismi di overloading e overriding dei metodi del C++.

2 (10 punti)

Si descriva il ruolo svolto dal polimorfismo in una gerarchia di classi, esaminando i meccanismi sintattici disponibili nel C++.

3 (10 punti)

Si descriva il meccanismo del passaggio di parametri per riferimento, considerando anche il caso in cui si usi il qualificatore `const`. Una funzione o metodo può restituire un riferimento ad un oggetto?

4 (3 punti)

Sia data una gerarchia di classi che discende da Shape, tra cui Rectangle e Triangle che specializzano il metodo `draw()`.

```
class Shape {
public:
     Shape();
     virtual ~Shape();
     virtual void draw() = 0;
};
```

Completare il seguente programma, in corrispondenza del FIXME:

```
#include <iostream>
#include <vector>
#include "Shape.h"
#include "Triangle.h"
#include "Rectangle.h"
using namespace std;

int main() {

     vector<Shape*> shapes;
     Triangle* aT = new Triangle;
     Rectangle* aR = new Rectangle;

     shapes.push_back(aT);
     shapes.push_back(aR);
     vector<Shape*>::const_iterator itr;
     for (itr = shapes.begin(); itr != shapes.end(); itr++ )
          .........; // FIXME: invocare il metodo draw sull'oggetto
                     // selezionato dall'iteratore

     return 0;
}
```

5 (8 punti)

Scrivere un costruttore di copia, distruttore ed un operatore di assegnazione per la seguente classe.

```
class MyString {
public:
	MyString(int size) {
		_size = size;
		_st = new char[_size];
	};
	virtual ~MyString() {
		// FIXME scrivere il distruttore !
	};
	MyString& operator=( // FIXME scrivere come passare i parametri ) {
		// FIXME scrivere il codice dell'operatore
	};
	void setCharAt(int pos, char value) {
		if( pos>=0 && pos < _size )
			_st[pos] = value;
	};
	char getCharAt(int pos){
		if( pos>=0 && pos < _size )
			return _st[pos];
		else
			return '\0';
	};
	int getSize() { return _size; };

private:
	char* _st;
	int _size;
};
```

6 (9 punti)

Si definisca una classe template che rappresenta un'immagine bidimensionale. Le immagini devono avere un numero variabile di bande (es. l'utente userà 1 per gray level, 3 per RGB, etc.), di righe e di colonne. Il parametro del template deve consentire di cambiare il tipo usato per la rappresentazione dei pixel (es. unsigned char, int, etc.). Le dimensioni dell'immagine (X ed Y) ed il numero di bande sono passate nel costruttore, insieme con un valore di inizializzazione dell'immagine. La classe ha i metodi generici getPixel e setPixel che rendono e impostano il valore di un pixel ad una data coordinata.

7 (10 punti)

Si scriva un Object Adapter per la seguente classe RGBPixel per fare in modo che possa essere usata con un client che si aspetta l'interfaccia definita nella classe GrayPixel. Se ne disegni il diagramma UML di classe.
Per calcolare la luminosità di un pixel, a partire dalla tripletta RGB corrispondente, si usi la formula:
grigio = 0.299 * R + 0.587 * G + 0.114 * B

```
class RGBPixel {
public:
     RGBPixel() { _R = _G = _B = 100; };
     int getR() { return _R; };
     int getG() { return _G; };
     int getB() { return _B; };
     void setR(int R) { _R = R; };
     void setG(int G) { _G = G; };
     void setB(int B) { _B = B; };

private:
     int _R;
     int _G;
     int _B;
};

class GrayPixel {
public:
     virtual unsigned char getPixelValue() = 0; // return gray level
};
```