Laboratorio Tecnologie dell' Informazione

PROVA SCRITTA – 23 Luglio 2009
Parte I – TEORIA

1 (10 punti)

Discutere lo scopo della programmazione generica. Descrivere come questa è implementata in C++.

2 (10 punti)

Si descriva l'uso della creazione dinamica di oggetti in C++, descrivendo anche l'ordine di chiamata a costruttori e distruttori in una gerarchia di classi.

3 (10 punti)

Si descriva il meccanismo di ereditarietà multipla del C++, considerando anche il caso della classe base duplicata (problema del diamante).

4 (12 punti)

Implementare tutti i metodi della seguente classe, completando anche le dichiarazioni dei metodi (es. i parametri). Il costruttore di copia e l'operatore di assegnazione devono usare un metodo privato di utilità che effettua la copia vera e propria.

```
class IntArray {
public:
      // FIXME costruttore con argomento con valore di default
      // l'array deve essere creato con una dimensione di 10 elementi, se
      // la dimensione non e' specificata dall'utilizzatore della classe
      IntArray(unsigned int ... ... ) {
            // FIXME evitare l'allocazione di array di 0 elementi,
            // impostare la dimensione almeno ad 1
      }
      // Copy constructor
      IntArray(...) {
            // FIXME usare copyInto()
      }
      // distruttore
      ~IntArray() {
            // FIXME scrivere distruttore
      }
      // operatore [] per accesso ad elementi array
      int& operator[](unsigned i) {
            // FIXME se si cerca di accedere ad un elemento fuori range
            // si deve riallocare l'array con una dimensione doppia dell'attuale
            // FIXME usare il metodo privato resize()
      }
      // operatore di assegnazione
      IntArray& operator=( ... ... ) {
            // FIXME usare copyInto()
      }
      // elementi dell'array usati
      unsigned size() const {
            // FIXME completare
      }
      // totale degli elementi dell'array
      unsigned capacity() const {
            // FIXME completare
      }

private:
      int* _buffer;
      unsigned _capacity;      // dimensione allocata al buffer
      unsigned _used;   // numero di elementi usati
```

```
	// metodo di utilita' per costruttore e operatore copia
	void copyInto(const IntArray& b) {
		// FIXME scrivere
	}
	// metodo di utilita'
	void resize(unsigned int s) {
		// FIXME ridimensionare e copiare i dati dell'array
	}
};
```

5 (18 punti)

Si completi la classe MyImage e dalla classe DisplayImage si derivino due classi, una che mostra l'immagine (basta un `cout`, chiamare la classe DisplayImageContent) ed una che mostra i metadati (usare `cout` per stampare i dati dell'immagine, chiamare la classe DisplayImageMetadata).
Usare il design pattern Observer (con approccio di tipo push) per fare in modo che ogni cambiamento apportato all'immagine sia immediatamente mostrato dalle classi responsabili del suo display.
Si disegni il diagramma UML delle classi coinvolte.

```
class MyImage {
public:
	MyImage() : _X(100), _Y(100), _channels(3), _transp(1.0) {};
	void setParams(int X, int Y, int channels, float trasp) {
		// FIXME completare anche il metodo
	};

private:
	int _X, _Y; // dimensioni immagine, devono assumere SOLO valore >0
	int _channels; // canali dell'immagine: 1 = gray, 3 = RGB, etc.,
			// deve assumere SOLO valori > 0
	float _transp; // livello di trasparenza dell'immagine,
			// deve assumere SOLO valori tra 0 e 1
};

class DisplayImage {
public:
	virtual void display() = 0; // mostra dati immagine
	virtual ~DisplayImage() = 0;
};
```