Laboratorio Tecnologie dell' Informazione

PROVA SCRITTA – 10 Settembre 2009
Parte I – TEORIA

1 (10 punti)

Si descriva il meccanismo di run-time type identification (RTTI) del C++, indicando quando si può usare. Si descriva l'uso dell'operatore `dynamic_cast<>`.

2 (10 punti)

Si descrivano gli elementi principali della Standard Template Library (STL), indicando quali relazioni hanno tra di loro. Scrivere un semplice esempio di uso di iteratore per accedere agli elementi di un vettore.

3 (10 punti)

Indicare cos'è un riferimento in C++, evidenziando le differenze rispetto ad un puntatore. E' possibile fare in modo che una variabile riferimento venga riassegnata per farla riferire ad un nuovo oggetto ? Scrivere un esempio di passaggio per riferimento di un parametro di una funzione.

4 (7 punti)

Completare la seguente classe scrivendo una dichiarazione di metodo "void fight()" in modo tale che risulti astratta. Estendere la classe creando due sottoclassi polimorfiche Elf e Dwarf che specializzano tale metodo.

```
class Player {
public:
     Player(string name); // FIXME: scrivere costruttore
     virtual ~Player();
     ................;
private:
     string _name; // FIXME: scrivere metodo getter del nome
};
```

Le sottoclassi devono poi essere utilizzabili nel seguente programma, che deve essere completato nelle righe indicata da FIXME:

```
#include <iostream>
#include <vector>
#include "Player.h"
#include "Elf.h"
#include "Dwarf.h"
using namespace std;

int main() {

     vector<Player*> playerGroup;
     Elf* aElf = new Elf("Legolas");
     Dwarf* aDwarf = new Dwarf("Gimli");

     playerGroup.push_back(aElf);
     playerGroup.push_back(aDwarf);
     vector< Player *>::const_iterator itr;
     for (itr = playerGroup.begin(); itr != playerGroup.end(); itr++ ) {
          cout << ..... ...; // FIXME: stampare il nome del personaggio
          ..... ...; // FIXME: come invoco questo fight ?
     }

     return 0;
}
```

5 (8 punti)

Date le seguenti classi e strutture aggiungere un metodo getAnimals alla classe Zoo, tale che renda possibile leggere il contenuto del vettore _animals garantendo che questo vettore non possa essere modificato e che l'operazione sia veloce anche quando il vettore contiene molti dati.

```
#include <vector>

struct Animal
{
	int _X; // non importa cos'e' _X
};

class Zoo
{
public:
	// FIXME scrivere il metodo getAnimals che legge senza modificare
il vettore _animals

private:
	std::vector<Animal> _animals;
};
```

Il metodo deve essere scritto in modo tale che la seguente istruzione non sia neanche compilabile:

```
zoo.getAnimals()[0]._X = 123; // non deve neanche compilare !
```

6 (15 punti)

Si consideri la seguente classe NetworkOperation, che mantiene lo stato di un'operazione di copia di un certo numero di file (_total) attraverso una rete. Si scrivano due classi: NetworkOperationViewer e NetworkOperationPercentViewer che mostrano lo stato delle operazioni di copia dei file, rispettivamente come numero di file copiati sul totale e come percentuale. Si adotti il pattern Observer per gestire questa funzione, modificando adeguatamente anche la classe NetworkOperation e si disegni il diagramma UML di classe. Si scriva un programma che, sfruttando queste classi, inizializzi un oggetto NetworkOperation (impostandone il totale dei file da copiare a 15) e creando due viewer (uno di ogni tipo). Il programma deve anche simulare la copia dei file richiamando il metodo updateOperations.

```
class NetworkOperation {
public:
     NetworkOperation(int total) : _total(total) {};
     void updateOperations(int processed);
     int getCurrent() const { return _current; };
     int getTotal() const { return _total; };
private:
     int _total;
     int _current;
};
```