Esempi di domande di compiti del corso “Programmazione”

Conoscenza del C++ e OOP

1 (12 punti)

Si descrivano i meccanismi di implementazione del polimorfismo nel C++, considerando anche il caso di costruttori e distruttori. (9 punti)
Un metodo virtuale può essere statico? Motivare la risposta. (3 punti)

(14 punti)

Si discuta il meccanismo delle eccezioni del C++, indicando come lanciare e gestire un’eccezione. (8 punti)
Un costruttore può lanciare un’eccezione ? Motivare la risposta. (3 punti)
Un distruttore può lanciare un’eccezione ? Motivare la risposta. (3 punti)

(14 punti)

Si discuta l’implementazione del paradigma di programmazione generica del C++, riferendosi all’implementazione di funzioni e classi (7 punti). Indicare come strutturare il codice nei file sorgenti (2 punti) e discutere il meccanismo della specializzazione e dei requisiti impliciti (5 punti).

(10 punti)

Discutere i principi alla base della programmazione Object Oriented (Data abstraction, ereditarietà e polimorfismo) e come questi siano implementati nel linguaggio C++.

(12 punti)

Si discuta la const correctness, con particolare riferimento ai puntatori e riferimenti del C++, oltre che ai metodi costanti.

(10 punti)

Descrivere gli smart pointer del C++, con particolar riferimento a unique e shared pointer. Come si usano ?

Esercizi di programmazione

(14 punti)

Implementare una classe template per la rappresentazione di matrici bidimensionali.
Implementare costruttore per copia e operatore di assegnazione, oltre a metodi per impostare e leggere valori a determinate coordinate. Scrivere anche un costruttore che prenda in ingresso dimensioni e valore di default degli elementi della matrice.

(12 punti)

Si considerino le seguenti classi relative ai personaggi di un videogioco. Si scrivano le classi necessarie (Abstract Factory) a fare in modo che ad uno AssaultTrooper possa essere associato solo un AssaultRifle ed una Stamina, mentre ad uno Sniper siano associate solo SniperRifle e HoldBreath (10 punti). Si disegni il diagramma UML di classe (2 punti).

```
class Weapon {
public:
	virtual ~Weapon() = 0;
};
Weapon::~Weapon() {}

class AssaultRifle : public Weapon {

};

class SniperRfile : public Weapon {

};

class PhysicalFeature {
public:
	virtual ~PhysicalFeature() = 0;
};
PhysicalFeature::~PhysicalFeature() {}

class Stamina : public PhysicalFeature {

};

class HoldBreath : public PhysicalFeature {

};


class Character {
public:
	virtual ~Character() = 0;
```

```
      void setPhysical(PhysicalFeature * aPF) {
      pf = aPF; }
void setWeapon(Weapon* aW) {
      w = aW; }
protected:
      PhysicalFeature * pf;
      Weapon* w;
};
Character::~Character() {}

class AssaultTrooper : public Character {
public:
      AssaultTrooper (int s) :
            strength (s) {};
      void doRun() {};
private:
      int strength;
};

class Sniper : public Character {
public:
      Sniper (int p) :
            precision(p) {};
      void doSnipe() {};
private:
      int precision;
};
```

(18 punti)

È necessario scrivere un costruttore di copia ed un operatore di assegnazione per la seguente classe? Indicare perché e nel caso implementare i metodi necessari.
È necessario scrivere un distruttore ? Nel caso implementarlo. (12 punti)
L'implementazione della classe Buffer fornita nell'esercizio è a rischio di memory leak in caso di problemi. Riscrivere la classe usando l'idioma RAII (Resource Acquistion Is Initialization), e mostrare come usare la classe. (6 punti)

```
class Buffer {
public:
      Buffer(int size) : size(size) { }
      void allocate() {
            buffer = new char[size];
      }
      void setCharAt(int pos, char value) {
            if( pos>=0 && pos < size )
                  buffer[pos] = value;
      }
      char getCharAt(int pos){
            if( pos>=0 && pos < size )
                  return buffer[pos];
```

```
            else
                return '\0';
        }
        int getSize() const { return size; }

private:
        char* buffer;
        int size;
};
```

(12 punti)

Scrivere una classe che rappresenti un file con nome, estensione e date di creazione, modifica ed ultimo accesso (6 punti). La classe file deve includere un operatore <= che usa la data di creazione per il confronto. Non implementare un costruttore di default. La data deve includere giorno, mese e anno + ora, minuto e secondo; creare la classe data (3 punti).
Indicare se le due classi hanno bisogno di implementare il costruttore di copia, l'operatore di assegnazione ed un distruttore e motivare la risposta. (3 punti)

(12+8 punti)

Si consideri lo sviluppo di un sistema software in cui alcune componenti vecchie devono essere ancora usate senza poterle aggiornare ("legacy"). In questo sistema si abbia quindi la classe `LegacyElevator` che implementa del codice per muovere un ascensore; il metodo `move_by` indica di quanti piani spostarsi in alto o in basso (per es. si passa al metodo -2 per scendere di 2 piani, +3 per salire di 3 piani). Questo codice non può essere modificato.
La creazione di un nuovo software di controllo degli ascensori richiede però di implementare l'interfaccia `Elevator` descritta sotto, il cui metodo `move_to` deve ricevere come argomento il numero del piano a cui andare.
Si implementi un Adapter a scelta (10 punti). Per ottenere i 6 punti aggiuntivi si implementi la seconda versione di Adapter, diversa dalla precedente.
Per ogni Adapter implementato se ne disegni il diagramma UML di classe (2 punti ogni diagramma).

```
class LegacyElevator {
public:
    LegacyElevator() { }

    int get_current_level() const;
    void move_by(int level);
    void open_doors();
    void close_doors();
};


class Elevator {
public:
```

```
    virtual void move_to(int level) = 0;
    virtual void open_doors() = 0;
    virtual void close_doors() = 0;
    virtual ~Elevator() { }
};
```

(7 punti)

Implementare una classe che rappresenta un numero complesso usando double sia per la parte reale che di quella immaginaria. La classe deve avere l'operatore di uguaglianza (4 punti). Implementare anche gli operatori per somma e sottrazione (2 punti).
La classe ha bisogno dell'operatore di assegnazione e di copia profonda (*deep copy*) ? (1 punto)

```
class Complex {
public:
     // TODO
private:
     double re;
     double im;
};
```

(10 punti)

Si implementino le classi `TelephoneNumber`, `Address`, e `Person` che rappresentano numeri di telefono (prefisso e numero), indirizzo (tipo di indirizzo come Via, Piazza, etc., nome e numero civico) e persona (titolo, nome e cognome). Si crei a partire da queste classi la classe `Contact` che rappresenta una persona con relativi numeri di telefono e indirizzi, che possono essere di tipo casa e ufficio (es. il numero di telefono 555-666333 è un telefono di casa). Un Contact può avere più indirizzi e numeri di telefono sia di tipo diverso che uguale (es. due indirizzi di tipo ufficio). (2 punti ogni classe, tranne Contact che vale 4)