

## Programmazione

PROVA SCRITTA – 6 Luglio 2017  
Parte I – LINGUAGGIO C++

### 1 (12 punti)

Si descrivano i livelli di accesso nelle classi C++ e come questi cambino secondo il tipo di ereditarietà. Discutere le relazioni tra classi implementate dai diversi tipi di ereditarietà. Un costruttore può essere privato? Se sì, come può essere invocato?

### 2 (10 punti)

Discutere l'uso dei membri statici di una classe. Indicare come si dichiarano, definiscono e usano. Fare un esempio di una classe con attributo e metodo statico. Un metodo statico può accedere a un attributo non statico? Un metodo non statico può accedere ad un membro statico? Motivare le risposte.

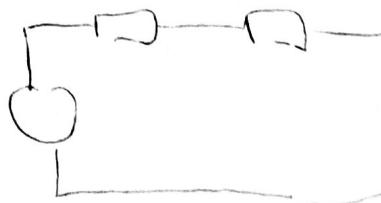
### 3 (8 punti)

Cos'è una reference e quando è bene usarla? Indicare le maggiori differenze rispetto all'uso dei puntatori. Modificare il seguente codice per far calcolare nel modo più veloce e sicuro possibile la somma degli elementi di un vettore STL:

```
int sum( /* FIXME */ ) {  
    int sum = 0;  
    for (/* FIXME */) {  
        // FIXME  
    }  
    return sum;  
}
```

```
std::vector<int> vec = {100, 200, 300};  
cout << "Sum: " << sum(/* FIXME vec*/) << endl;
```

~ 487,260 ~



## Programmazione

PROVA SCRITTA – 6 Luglio 2017  
Parte II – PROGRAMMAZIONE

### 4 (10 punti)

Sia data la classe AudioPlayer che implementa la interface class MediaPlayer; questa classe sa riprodurre file di tipo WAV ed è usata in un programma per l'ascolto di file audio. Allo scopo di aggiungere nuove capacità si decide di usare una libreria per la riproduzione di altri formati audio come MP3 e AAC. Le classi di questa libreria sviluppata da terze parti implementano però un'interface class diversa (AdvancedMediaPlayer). Implementare un Adapter per poter usare questa libreria (si imposti checkDRM a true). Disegnare lo schema UML (2 punti).

```
class MediaPlayer {
public:
    virtual void play(std::string audioType, std::string filename) = 0;
};

class AdvancedMediaPlayer {
public:
    virtual void play(std::string filename, bool checkDRM) = 0;
};

class AACPlayer : public AdvancedMediaPlayer {
public:
    virtual void play(std::string, bool checkDRM) override {
        std::cout << "AACPlayer playing..." << std::endl;
    }
};

class MP3Player : public AdvancedMediaPlayer {
public:
    virtual void play(std::string, bool checkDRM) override {
        std::cout << "MP3Player playing..." << std::endl;
    }
};

class AudioPlayer : public MediaPlayer {
public:
    virtual void play(std::string audioType, std::string filename) override {
        if (audioType != "wav")
            std::cout << "Unknown file format: " << audioType << std::endl;
        else
            std::cout << "AudioPlayer playing WAV..." << std::endl;
    }
};

int main() {
    AudioPlayer* ap = new AudioPlayer;
    ap->play("wav", "test.wav");
    ap->play("mp3", "test.mp3");
    ap->play("aac", "test.mp3");
}
```

## 5 (12 punti)

Si disegni e implementi una classe `DiskFile` che rappresenta un file di un filesystem di un sistema operativo con nome del file (ovvero contiene qualcosa del tipo "data.xls"), dimensione in byte e variabile che indica se scrivibile o no. Si disegni e implementi una classe `DiskDirectory` che rappresenta una directory di un filesystem di un sistema operativo con nome della directory (ovvero contiene qualcosa del tipo "Documents") e l'elenco di directory e file inclusi al suo interno. La classe `DiskDirectory` deve avere un metodo `list()` che stampa su schermo l'elenco di directory e file in essa contenuti. La classe deve avere un metodo `add()` per aggiungere directory e file, ed un metodo `remove()` che dato un nome cancella l'elemento corrispondente. Sia `DiskFile` che `DiskDirectory` devono avere un metodo `print()` che ne stampa i dettagli.

Dato che una directory non può contenere file o directory con lo stesso nome si consiglia l'uso di `template<class key_type, class mapped_type> std::map`, di cui si riportano i seguenti metodi utili allo sviluppo delle classi:

Inserimento e selezione di elementi:

```
mapped_type& operator[] (const key_type& k);
```

Ricerca di elementi (ritornano un iteratore a `map::end()` nel caso l'elemento cercato non sia presente):

```
iterator find (const key_type& k);
const_iterator find (const key_type& k) const;
```

Cancellazione di elementi:

```
void erase (iterator position);
size_type erase (const key_type& k);
void erase (iterator first, iterator last);
```

Si ricorda che gli iteratori di una mappa puntano agli elementi che sono di tipo `std::pair<const key_type, mapped_type>`, e che usando i membri `first` e `second` si ottengono la chiave e il valore cui puntano.

## 6 (10+5 punti)

Scrivere una classe generica che rappresenti un'immagine di larghezza W e altezza H, da specificare nel costruttore. L'immagine deve essere di tipo RGB, ovvero con tre valori (canali) per ognuno dei pixel, per cui il valore di ogni pixel dell'immagine è rappresentato da tre numeri. La classe deve fornire metodi per impostare e rendere il valore dei tre canali a determinate coordinate X e Y. Al momento della creazione tutti i canali di tutti i pixel devono essere impostati a 0 (8 punti). I metodi di impostazione e lettura del valore di un pixel devono lanciare le eccezioni `std::out_of_range` e `std::invalid_argument` rispettivamente nel caso le coordinate del pixel non siano accettabili o nel caso il valore per impostare il pixel sia negativo. (2 punti)

In caso sia necessario, implementare anche costruttore di copia e operatore di assegnazione. (5 punti)

## Programmazione

PROVA SCRITTA – 5 Settembre 2017  
Parte I – LINGUAGGIO C++

### 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)

### 2 (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)

### 3 (8 punti)

Si descrivano le caratteristiche e differenze di overloading e overriding di metodi in C++ (6 punti). Quando si ha il caso di hiding, ed a cosa serve la parola chiave override (2 punti)?

## Programmazione

PROVA SCRITTA – 5 Settembre 2017  
Parte II – PROGRAMMAZIONE

### 4 (8 punti)

Siano date le classi Widget, Button, TextArea e LayoutManager con Widget classe base astratta per le altre tre classi.

Modificare il seguente codice per implementare la specifica descritta sopra.

La classe LayoutManager deve poter contenere al suo interno qualsiasi classe derivata da Widget e quando se ne invoca il metodo draw() deve disegnare tutti gli oggetti contenuti.

```
class Widget {  
public:  
    void draw(); // TODO make abstract class  
};  
  
class Button {  
public:  
    Button(std::string t) : text(t) {}  
    // TODO implement draw()  
private:  
    std::string text;  
};  
  
class TextArea {  
public:  
    TextArea(std::string t) : text(t) {}  
    // TODO implement draw()  
private:  
    std::string text;  
};  
  
#include <list>  
class LayoutManager {  
public:  
    void draw() {  
        // FIXME  
        for()  
            // TODO draw all widgets  
    }  
    void addWidget(/* FIXME */) {  
        elements.push_back(/* FIXME */);  
    }  
    void removeWidget(/* FIXME */) {  
        elements.remove(/* FIXME */);  
    }  
};
```

```

    }
private:
    std::list< > elements; // FIXME can use any Widget
};

```

## 5 (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() { }
};

```

## 6 (10 punti)

Implementare una classe generica che rappresenta un pixel con 3 canali (R, G e B). La classe deve avere l'operatore di uguaglianza e disuguaglianza (5 punti). Implementare anche gli operatori per somma e sottrazione (4 punti).

La classe ha bisogno dell'operatore di assegnazione e di copia profonda (*deep copy*)?

Motivare la risposta. (1 punto)

## Programmazione

PROVA SCRITTA – 22 Giugno 2017  
Parte I – LINGUAGGIO C++

### 1 (10 punti)

Si descrivano gli speciatori di accesso ai membri di una classe e come i livelli di accesso varino secondo il tipo di ereditarietà in una classe derivata.

Indicare che tipo di relazione tra classi è indotta dalla ereditarietà di tipo public e motivare la risposta.

Indicare che tipo di relazione tra classi è indotta dalla ereditarietà di tipo private e motivare la risposta.

### 2 (12 punti)

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

### 3 (10 punti)

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

1 /

UPO  
Indirizzi  
name  
secret

## Programmazione

PROVA SCRITTA – 22 Giugno 2017  
Parte II – PROGRAMMAZIONE

### 4 (13 punti)

Implementare una classe `Date` che rappresenta una data del calendario gregoriano. La classe deve avere: 1) costruttore di default che imposta la data sul 1 Gennaio 1970; 2) metodi `getter` e `setter` per giorno, mese e anno; i metodi `setter` devono controllare che l'attributo che viene impostato mantenga una data corretta e in caso contrario devono lanciare l'eccezione `std::out_of_range` (es. se la data era 1 Febbraio 2017 e si cerca di impostare il giorno a 30 si deve lanciare l'eccezione). (10 punti) Si gestiscano gli anni bisestili: si ricorda che un anno è bisestile se è divisibile per 4 o per 400, ma non per 100: es. 1992 e 2000 sono bisestili, mentre 1997 e 1900 no. (3 punti)

### 5 (15 punti)

Scrivere le classi `Item`, `Customer` e `Order` che rappresentano rispettivamente: 1) un oggetto in vendita in un negozio (nome e prezzo in Euro, con indicazione anche dei centesimi) (3 punti); 2) un cliente rappresentato da nome, cognome, indirizzo e booleano che indica se cliente premium (2 punti); 3) un ordine composto da uno o più `Item`, che possono essere aggiunti o eliminati dall'ordine; l'ordine è associato ad un `Customer` ed ha un metodo che calcola il costo totale dell'ordine. Se il cliente è di tipo premium tutti i costi degli `Item` sotto i 15€ sono scontati del 10%.

### 6 (16 punti)

Si considerino le classi `Mail`, `Mailbox`, `GuiNotifier` e `IconBadgeMonitor` mostrate sotto. La classe `Mail` rappresenta un'email con titolo, mittente, corpo del testo e booleano che indica se la mail è stata già letta. `Mailbox` contiene le mail relative ad una cartella di posta di un programma di email, e deve consentire di aggiungere nuove email e leggere email ad una determinata posizione (es. la mail n°3), quindi ponendo a letto il booleano della `Mail` relativa. Le classi `GuiNotifier` e `IconBadgeMonitor` devono poter monitorare un oggetto di tipo `Mailbox` e ricevere informazioni sul cambiamento di stato di questo oggetto allo scopo, rispettivamente, di: 1) mostrare in un centro notifiche di un sistema operativo il titolo e mittente dell'ultima email arrivata; 2) mostrare il numero di email ancora da leggere disegnandolo sull'icona del programma di posta.

Si adotti il design pattern `Observer` (versione a scelta) per completare il codice (14 punti).

Si disegni il diagramma UML di classe della soluzione. (2 punti)

```
class Mail {  
public:  
    Mail(std::string ti, std::string se, std::string te, bool r=false);  
    std::string getTitle() const { return title; }  
    std::string getSender() const { return sender; }  
    std::string getText() const { return text; }  
    bool isRead() const { return read; }  
    void setRead(bool r) { read=r; }
```

```
private:
    std::string title, sender, text;
    bool read;
};

class Mailbox {
public:
    void addMail(const Mail& newMail) {
        emails.push_back(newMail);
    }
    const Mail& lastEmail() const {
        return emails.back();
    }
    void readEmail(int i) {
        if (i>=0 && i<emails.size()) {
            std::cout << emails[i].getTitle() << std::endl;
            std::cout << emails[i].getSender() << std::endl;
            std::cout << emails[i].getText() << std::endl;
            emails[i].setRead = true;
        } else
            std::cerr << "bad index" << std::endl;
    }
    int getNumUnreadEmails() const {
        // TODO count unread emails
    }
private:
    std::vector<Mail> emails;
};

class GuiNotifier {
public:
    GuiNotifier (bool act) : active(act) { };
    virtual ~GuiNotifier() {};
    // TODO implement draw+update method that prints notification
    // message if updated value (just use std::cout) and active
    // attribute is true. Add getter/setter methods "active" attribute
private:
    bool active;
};

class IconBadgeMonitor {
public:
    IconBadgeMonitor() { };
    virtual ~IconBadgeMonitor() {};
    // TODO implement draw+update method that prints changed
    // badge and updated value (just use std::cout)
};
```

## Programmazione

PROVA SCRITTA – 21 Settembre 2017  
Parte I – TEORIA

### 1 (10 punti)

Si descriva il meccanismo delle eccezioni in C++, descrivendo gli aspetti implementativi.

### 2 (10 punti)

Discutere le funzioni membro virtuali, la loro definizione e quali differenze hanno con metodi non virtuali. Considerare il caso dei distruttori virtuali e quando è necessario usarli. Indicare scopo e uso dei metodi puramente virtuali.

### 3 (10 punti)

Si descriva l'idioma RAI, discutendo in quali casi risulti utile (4 punti). Si descrivano quindi gli smart pointer del C++, con particolar riferimento a unique e shared pointer (5 punti). Come si usano (1 punto) ?

## Programmazione

PROVA SCRITTA – 21 Settembre 2017  
Parte II – PROGRAMMAZIONE

### 4 (10+2 punti)

Si considerino le classi PacketMon e CPUUsage mostrate sotto, assieme alla classe MultiMonitor. La classe MultiMonitor deve poter monitorare sia un oggetto di tipo PacketMon che uno di tipo CPUUsage (es. per implementare un display di sistema di un sistema operativo), e ricevere informazioni sul cambiamento di stato di ognuno di questi oggetti, allo scopo di mostrarlo. Si adotti il design pattern Observer (si consiglia nella versione pull) per completare il codice, e si disegni il diagramma UML di classe di questo problema. Se necessario si sfrutti l'RTTI per invocare i metodi specifici dei soggetti da monitorare per ottenerne lo stato.

Bonus: il packet monitor avvisa quando il numero di pacchetti è divisibile per 100, CPU usage avvisa quando la percentuale di uso è divisibile per 10 (2 punti).

```
class PacketMon {
public:
    PacketMon() : packets(0) { }
    void addPacket(int packets) {
        this->packets += packets;
    }
    int getPackets() const { return packets; }
private:
    int packets;
};

class CPUUsage {
public:
    CPUUsage() : use(0) { }
    void update(int howMuch = 1) {
        use += howMuch;
    }
    int getUsage() const { return use; }
private:
    int use;
};

class MultiMonitor {
public:
    MultiMonitor(string monitorName) : name(monitorName) { }
    virtual ~MultiMonitor() {}
    // TODO implement update method that prints monitor name
    // and updated value of packet or CPU usage
private:
    string name;
};
```

## 5 (10 punti)

Si consideri il seguente frammento di codice e si riscriva in modo tale da evitare che un'eccezione lanciata prima del delete provochi un leak. Scrivere una soluzione usando la gestione di eccezioni (3 punti) ed una usando uno smart pointer (2 punti).

```
void test()
{
    Resource *ptr = new Resource;
    // do something that may throw an exception
    ptr->update();
    // ...
    delete ptr;
```

Supponendo che la classe Resource sia:

```
class Resource
{
public:
    Resource() : dvc(new std::device) {}
    void update() throw(std::runtime_error);
    // other methods...
private:
    Device *const dvc;
};
```

e usi

```
class Device
{
    // something...
};
```

modificare la classe Resource per implementare l'idioma RAII (3 punti). Dare un'implementazione anche con l'uso di smart pointer (2 punti).

## 6 (10 punti)

Si definiscono due classi: una che rappresenta un pixel nello spazio di colore RGB (ovvero usando 3 valori interi rappresentati con almeno 10 bit, uno per ogni canale di colore R, G e B) ed una che rappresenta un pixel nello spazio di colore a livelli di grigio (1 valore intero che può variare tra 0 e 255) (2 punti).

Si definisca una classe template che rappresenta un'immagine bidimensionale, le cui dimensioni (larghezza ed altezza) sono passate nel costruttore. La classe è quindi composta da larghezza x altezza pixel. La classe ha i metodi generici getPixel e setPixel che rendono e impostano il valore di un pixel ad una data coordinata. (8 punti)

## Programmazione

PROVA SCRITTA – 19 Luglio 2017  
Parte I – TEORIA

### 1 (12 punti)

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

### 2 (8 punti)

Si discutano i modi con cui si passano i parametri di funzioni e metodi in C++ (8 punti). Si scriva una funzione che ritorna il massimo di un vettore STL di interi.

### 3 (6 punti)

Si descrivano le caratteristiche e differenze di overloading e overriding di metodi in C++. Quando si ha il caso di hiding ?

### 4 (10 punti)

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

## Programmazione

PROVA SCRITTA – 19 Luglio 2017  
Parte II – PROGRAMMAZIONE

### 5 (12 punti)

Si considerino le seguenti tre classi `FileSystemObj`, da cui sono derivate `File` e `Directory`. A partire di `File` e `Directory` si creano due nuove specializzazioni per Windows e Linux (i.e. `LinuxFile` e `LinuxDirectory`, etc.). Il metodo `isHidden()` della versione Linux rende vero se il primo carattere del nome è "", mentre la versione per Windows controlla un attributo booleano da aggiungere. Il metodo `list()` per Windows stampa a schermo i file dal primo all'ultimo, mentre la versione Linux dall'ultimo al primo. Si implementi un Abstract Factory, disegnandone il diagramma UML di classe (2 punti), per la creazione delle famiglie di prodotti file e directory. Completare, dove richiesto, il programma `main.cpp`.

```
class FileSystemObj {  
public:  
    virtual bool create(string aName) = 0;  
    virtual bool move(string newName) = 0; // set new name  
public:  
    string name;  
};  
  
class File : public FileSystemObj {  
public:  
    virtual bool isHidden() const = 0;  
    int numBytes;  
};  
  
class Directory : public FileSystemObj {  
public:  
    void addFile(File newFile) {  
        files.push_back(newFile);  
    }  
    void removeFile(File oldFile) {  
        files.remove(oldFile);  
    }  
    virtual void list() = 0;  
private:  
    std::list<File> files;  
};
```

```
// main.cpp
// TODO add includes
using namespace std;

int main() {
    FSFactory* factory;
    bool isWindows;

    // set isWindows value here

    if( isWindows ) {
        // TODO add code to create correct factory here
    } else {
        // TODO add code to create correct factory here
    }

    // TODO add code here to create handlers for a
    // file and for a directory object

    file->create("test.txt");
    dir->create("/tmp/foo");
    dir->addFile(*file);
    dir->move("/tmp/bar");

    return 0;
}
```

## 6 (12 punti)

Si implementi una classe template Backpack. Il costruttore della classe consente di indicare la dimensione dello zaino (ovvero il volume a disposizione). La classe ha un metodo add che riceve in ingresso un elemento di tipo uguale al template da aggiungere allo zaino, ed un intero che indica lo spazio occupato. Il metodo rende vero se c'è abbastanza spazio nello zaino da consentire l'inserimento (e l'inserimento riduce lo spazio disponibile) e falso altrimenti (l'oggetto non è inserito). La classe ha anche un metodo remove che toglie l'ultimo elemento inserito nello zaino e aggiorna lo spazio disponibile.

## **7 (12 punti)**

Scrivere una classe Book che rappresenti un libro (titolo, autore, editore, numero di pagine), ed una classe DVD che rappresenti un DVD con film (titolo, regista, durata in minuti). (3 punti)

Scrivere una classe Library che contenga in una lista sia DVD che Book, consenta l'inserimento ed eliminazione di questi oggetti e stampi l'intero contenuto della libreria, con i dettagli dei vari elementi (6 punti).

Indicare se le tre classi hanno bisogno di implementare il costruttore di copia, l'operatore di assegnazione ed un distruttore e motivare la risposta. (3 punti)

(17)

(16)

## Programmazione

PROVA SCRITTA – 19 Febbraio 2018  
Parte I – TEORIA

### 1 (12 punti)

Si descrivano i meccanismi di implementazione del polimorfismo nel C++, considerando anche il caso di costruttori e distruttori. (9 punti)  
Un costruttore può essere virtuale? Un distruttore? Motivare la risposta. (3 punti) (7)

### 2 (12 punti)

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

(3)

### 3 (8 punti)

Si descrivano le caratteristiche e differenze di overloading e overriding di metodi in C++ (6 punti). Quando si ha il caso di hiding, ed a cosa serve la parola chiave override (2 punti)? ✓

(6)

(5)

20  
**Programmazione**

PROVA SCRITTA – 19 Febbraio 2018  
Parte II – PROGRAMMAZIONE

**4 (8 punti)**

Siano date le classi Widget, Button, TextArea e LayoutManager con Widget classe base astratta per le altre tre classi.

Modificare il seguente codice per implementare la specifica descritta sopra.

La classe LayoutManager deve poter contenere al suo interno qualsiasi classe derivata da Widget e quando se ne invoca il metodo draw() deve disegnare tutti gli oggetti contenuti.

```
class Widget {
    public:
        void draw(); // TODO make abstract class
};

class Button {
    public:
        Button(std::string t) : text(t) {}
        // TODO implement draw()
    private:
        std::string text;
};

class TextArea {
    public:
        TextArea(std::string t) : text(t) {}
        // TODO implement draw()
    private:
        std::string text;
};

#include <list>
class LayoutManager {
    public:
        void draw() {
            // TODO iterate elements and draw them
        }
        void addWidget(Widget* w) {
            elements.push_back(w);
        }
        void removeWidget(Widget* w) {
            elements.remove(w);
        }
    private:
        std::list<Widget> elements; // FIXME can use Widget
};
```

(P)

## 5 (10 punti)

Un programma per la gestione di contatti deve poter rappresentare una persona con sue proprietà di indirizzo, numero di telefono ed email. Ognuna di queste proprietà deve essere caratterizzabile con un tipo (es. email di lavoro, indirizzo di casa, telefono cellulare, telefono lavoro, etc.)

Una persona può avere più proprietà (es. più indirizzi, sia di tipi diversi che uguali). (2)

Disegnare e implementare le classi necessarie per la rappresentazione di un contatto.

## 6 (12+8 punti)

Sia data la classe VideoPlayer che implementa la interface class MediaPlayer; questa classe sa riprodurre file di tipo AVI ed è usata in un programma per visualizzare file video. Allo scopo di aggiungere nuove capacità si decide di usare una libreria per la riproduzione di altri formati audio come H.264 e WMV. Le classi di questa libreria sviluppata da terze parti implementano però un'interface class diversa (AdvancedMediaPlayer). Implementare un Adapter per poter usare questa libreria (si imposta checkDRM a true). Disegnare lo schema UML (2 punti).

```
class MediaPlayer {  
public:  
    virtual void play(std::string videoType, std::string filename) = 0;  
};  
class AdvancedMediaPlayer {  
public:  
    virtual void videoplay(std::string filename, bool checkDRM) = 0;  
};  
class WMVPlayer : public AdvancedMediaPlayer {  
public:  
    virtual void videoplay(std::string, bool checkDRM) override {  
        std::cout << "WMVPlayer playing..." << std::endl;  
    }  
};  
class H264Player : public AdvancedMediaPlayer {  
public:  
    virtual void videoplay(std::string, bool checkDRM) override {  
        std::cout << "H264Player playing..." << std::endl;  
    }  
};  
class VideoPlayer : public MediaPlayer {  
public:  
    virtual void play(std::string audioType, std::string filename) override {  
        if (audioType != "avi")  
            std::cout << "Unknown file format: " << audioType << std::endl;  
        else  
            std::cout << "VideoPlayer playing AVI..." << std::endl;  
    }  
};  
  
int main() {  
    VideoPlayer* vp = new VideoPlayer;  
    vp->play("avi", "test.avi");  
    vp->play("h264", "test.h264");  
    vp->play("wmv", "test.wmv");  
}
```

## 7 (7 punti)

Implementare una classe generica che rappresenta coordinate cartesiane bidimensionali.  
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*)?  
Motivare la risposta. (1 punto)

## Programmazione

PROVA SCRITTA – 18 Giugno 2018  
Parte I – LINGUAGGIO C++ e OOP

### 1 (12 punti)

Si descrivano i livelli di accesso nelle classi C++ e come questi cambino secondo il tipo di ereditarietà. (8 punti)

Un costruttore può essere privato? Se sì, come si invoca? Fornire un esempio di codice (4 punti)

### 2 (14 punti)

Si discuta funzionamento e uso dell'idioma RAI in C++ (8 punti).

Discutere funzionamento, uso e differenze tra `unique_ptr<>` e `shared_ptr<>` (6 punti).

### 3 (10 punti)

Descrivere il meccanismo delle eccezioni nel C++: come si dichiarano, lanciano, prendono (anche in relazione all'ereditarietà) (8 punti). Spiegare le maggiori differenze tra uso di eccezioni e l'uso di valori di ritorno per indicare la presenza di errori a *runtime*. (2 punti)

**Programmazione**PROVA SCRITTA – 18 Giugno 2018  
Parte II – PROGRAMMAZIONE

(7)

**4 (8 punti)**

Siano date le classi (`DocumentElement`, `ImageElement`, `TextElement`) e `LayoutElement`. Modificarle per fare in modo che `DocumentElement` sia classe base astratta per tutte le altre tre classi.

base

La classe `LayoutElement` deve poter contenere al suo interno qualsiasi classe derivata da `DocumentElement` e quando se ne invoca il metodo `draw()` deve disegnare tutti gli oggetti contenuti.

```

class DocumentElement {
public:
    void draw(); // TODO make abstract class
private:
    int posX, posY;
};

class ImageElement {
public:
    ImageElement(std::string f) : fileName(f) {} // TODO implement draw()
    std::string fileName; // image file name
    int width, height;
};

class TextElement {
public:
    TextElement(std::string t) : text(t) {} // TODO implement draw()
private:
    std::string text;
    int fontSize;
};

#include <list>
class LayoutElement {
public:
    void draw() {
        // TODO iterate elements and draw them
    }
    void addElement(/* FIXME */) {
        elements.push_back(/* FIXME */);
    }
};

```

```

void removeElement(/* FIXME */) {
    elements.remove(/* FIXME */);
}
private:
    std::list</* FIXME */> elements; // document elements
};

```

## 5 (10 punti)

(5)

Allo scopo di implementare un programma di slideshow (presentazione di documenti multimediali) si implementino le seguenti classi:

- Playlist: contiene l'elenco ordinato dei documenti multimediali da presentare (immagini e video), indicando per ogni documento per quanti secondi questo deve apparire. Deve essere possibile associare un file audio da suonare durante la presentazione ed impostare la presentazione per avere una ripetizione.
- Image: contiene indicazione su nome del file e dimensioni dell'immagine.
- Video: contiene indicazioni su nome del file, dimensioni dei fotogrammi e durata.

## 6 (14 punti)

(6)

Si considerino le classi Mail, Mailbox, GuiNotifier e IconBadgeMonitor mostrate sotto. La classe Mail rappresenta un'email con titolo, mittente, corpo del testo e booleano che indica se la mail è stata già letta. Mailbox contiene le mail relative ad una cartella di posta di un programma di email, e deve consentire di aggiungere nuove email e leggere email ad una determinata posizione (es. la mail n° 3), quindi ponendo a letto il booleano della Mail relativa. Le classi GuiNotifier e IconBadgeMonitor devono poter monitorare un oggetto di tipo Mailbox e ricevere informazioni sul cambiamento di stato di questo oggetto allo scopo, rispettivamente, di: 1) mostrare in un centro notifiche di un sistema operativo il titolo e mittente dell'ultima email arrivata; 2) mostrare il numero di email ancora da leggere disegnandolo sull'icona del programma di posta.

Si adotti il design pattern Observer (versione a scelta) per completare il codice (12 punti). Si disegni il diagramma UML di classe della soluzione. (2 punti)

```

class Mail {
public:
    Mail(std::string ti, std::string se, std::string te, bool r=false);
    std::string getTitle() const { return title; }
    std::string getSender() const { return sender; }
    std::string getText() const { return text; }
    bool isRead() const { return read; }
    void setRead(bool r) { read=r; }

private:
    std::string title, sender, text;
    bool read;
};

```

```
class Mailbox {
public:
    void addMail(const Mail& newMail) {
        emails.push_back(newMail);
    }
    const Mail& lastEmail() const {
        return emails.back();
    }
    void readEmail(int i) {
        if (i>=0 && i<emails.size()) {
            std::cout << emails[i].getTitle() << std::endl;
            std::cout << emails[i].getSender() << std::endl;
            std::cout << emails[i].getText() << std::endl;
            emails[i].setRead = true;
        } else
            std::cerr << "bad index" << std::endl;
    }
    int getNumUnreadEmails() const {
        // TODO count unread emails
    }
private:
    std::vector<Mail> emails;
};

class GuiNotifier {
public:
    GuiNotifier (bool act) : active(act) { };
    virtual ~GuiNotifier() {};
    // TODO implement draw+update method that prints notification
    // message if updated value (just use std::cout) and active
    // attribute is true. Add getter/setter methods "active" attribute
private:
    bool active;
};

class IconBadgeMonitor {
public:
    IconBadgeMonitor() { };
    virtual ~IconBadgeMonitor() {};
    // TODO implement draw+update method that prints changed
    // badge and updated value (just use std::cout)
};
```

## Programmazione

PROVA SCRITTA – 4 Luglio 2018  
Parte I – LINGUAGGIO C++ e OOP

### 1 (8 punti)

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

### 2 (6 punti)

Discutere dei membri statici di una classe. Come si dichiarano, definiscono e usano. Fare un esempio di una classe con attributo e metodo statico. Un metodo statico può accedere a un attributo non statico (motivare la risposta)?

### 3 (6 punti)

Si descrivano le caratteristiche e differenze di overloading e overriding di metodi in C++. Quando si ha il caso di hiding ?

### 4 (10 punti)

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

**Programmazione**PROVA SCRITTA – 4 Luglio 2018  
Parte II – PROGRAMMAZIONE**5 (12 punti) (7)**

Si considerino le due classi base astratte GameCharacter e Weapon e se ne derivino le classi JediKnight, RebelGuard e StormTrooper (da GameCharacter), Blaster e LaserSword (da Weapon).

Si implementi un Abstract Factory, disegnandone il diagramma UML di classe (2 punti), per la creazione dei personaggi di un gioco in modo tale che solo i personaggi JediKnight abbiano una spada laser, mentre gli altri abbiano un Blaster (con strength 6 per RebelGuard e strength 8 per StormTrooper) come armi (8 punti). Scrivere un frammento di codice che crei un personaggio di ogni tipo (2 punti).

```
class GameCharacter {  
public:  
    virtual int fight(GameCharacter& enemy) = 0;  
    virtual void move(int x, int y) = 0;  
public:  
    Weapon* w;  
    int posX, posY;  
}  
  
class Weapon {  
public:  
    virtual int use(GameCharacter& enemy) = 0;  
public:  
    int strength;  
    bool shield;  
}
```

## 6 (10 punti)

Si consideri la struct Articolo che rappresenta un oggetto da acquistare rappresentato da un nome e quantità. Si scriva una classe che rappresenta una lista della spesa con i seguenti metodi: printAll() che stampa tutti gli elementi presenti, indicando nome e quantità, add() che aggiunge un articolo e nel caso sia già presente ne modifica la quantità secondo quanto indicato dall'elemento aggiunto; remove() che cancella un articolo dato il suo nome, getQuantity che riporta la quantità di un articolo sulla base del suo nome. (6 punti)

```
struct Articolo {  
public:  
    std::string nome;  
    int qty; // quantità  
};
```

- Nel caso si usi una `std::map<Key K, MappedType T>` si ricorda la presenza dei seguenti metodi:

`T& operator[]( const Key& key );` ritorna un riferimento al valore mappato sulla chiave equivalente a quella passata come argomento, effettuando un inserimento se tale chiave non esiste.

`iterator find( const Key& key );` cerca un elemento con chiave equivalente a quella indicata; rende l'iteratore all'elemento se presente o all'iteratore prima posizione dopo la fine se non presente (`end()`).

`void erase( iterator pos );` rimuove l'elemento alla posizione indicata dall'iteratore

e che i valori presenti nella mappa sono del tipo: `std::pair<const Key, T>` con attributi pubblici `first` per la chiave e `second` per il valore associato.

- Nel caso si usi una `std::list<ValueType T>` si ricorda la presenza dei seguenti metodi:

`void push_back( const T& value );` aggiunge l'elemento in coda alla lista

`iterator erase( iterator pos );` rimuove l'elemento alla posizione indicata dall'iteratore

e l'algoritmo `find`:

```
template< class InputIt, class T >  
InputIt find( InputIt first, InputIt last, const T& value );  
che cerca nell'intervallo tra l'iteratore first ed il last un valore. L'algoritmo usa  
l'operatore == che deve essere definito per il tipo su cui si applica.
```

Si ricorda che la firma di tale operatore è:

```
bool operator==(const T & right) const
```

12

## 7 (18 punti)

Scrivere una classe che rappresenti un file con nome, estensione e date di creazione, modifica ed ultimo accesso (6 punti). La data deve includere giorno, mese e anno + ora, minuto e secondo; creare la classe data (3 punti).

Implementare una classe directory con nome, data di creazione, modifica e ultimo accesso. Una directory deve contenere una collezione di file e di altre directory (6 punti) Indicare se le tre classi hanno bisogno di implementare il costruttore di copia, l'operatore di assegnazione ed un distruttore e motivare la risposta. (3 punti)

## Programmazione

PROVA SCRITTA – 18 Luglio 2018  
Parte II – PROGRAMMAZIONE

### 4 (10 punti)

Sia data una struct Pixel che rappresenta i valori R, G e B di un pixel di un'immagine.  
Sia data la seguente classe Image che rappresenta un'immagine. Il metodo setPixel imposta il valore di un pixel alle coordinate (x,y) modificando adeguatamente il contenuto del buffer dei Pixel (implementare il metodo), ed un metodo getPixel che ritorna il valore di un pixel alle coordinate richieste (implementare il metodo).

La classe ha un metodo save() che salva sul file dal nome specificato l'immagine, ed un metodo load() che legge un'immagine da file; questi metodi usano quattro funzioni che fanno parte di una qualche libreria per la lettura e scrittura di file di immagini.

La classe ha anche un metodo computeHash() che calcola una firma univoca che riassume il contenuto dell'immagine sotto forma di una stringa, calcolata a partire dal contenuto presente nel buffer in quel momento (si assuma che l'algoritmo che esegue questa operazione garantisce di ottenere valori diversi nella stringa per buffer che differiscono anche solo di un bit). Si tenga conto che l'esecuzione di questo metodo è particolarmente onerosa dal punto di vista computazionale per cui è bene eseguirlo solo quando necessario.

Completare la classe per fare in modo che descriva un'immagine, con dimensione orizzontale e verticale, e indicatore booleano che indica se l'immagine deve essere ancora salvata (es. perché non è ancora mai stata salvata o perché il contenuto è stato modificato rispetto all'ultimo salvataggio). Definire gli operatori == e != per comparare due immagini sulla base del valore calcolato con computeHash(); la soluzione deve efficiente dal punto di vista computazionale.

```
class Image {  
public:  
    .....;  
    void load(std::string name) {  
        buffer = decompressImage(name);  
    }  
    void save(std::string name) {  
        Pixel* compressedBuffer = compressBuffer(buffer);  
        saveCompressedBuffer(name, compressedBuffer);  
        freeBuffer(compressedBuffer);  
    }  
    void setPixel(int x, int y, const Pixel& pix);  
    Pixel getPixel(int x, int y);  
  
protected:  
    std::string computeHash() {  
        std::string result = "HashValue";  
        // slow computation returning a string  
        return result;  
    }  
private:  
    Pixel* buffer;  
};  
bool saved;  
int height, width;
```

## 5 (12 punti)

Si implementi una classe `AudioTrack` che rappresenta una canzone con titolo, durata in secondi e nome del file dell'immagine della copertina dell'album di cui fa parte. La classe ha un metodo `play()` che suona la canzone.

Si implementi quindi una classe `Playlist`, che include un elenco ordinato di `AudioTrack` da suonare in sequenza; la playlist ha un nome ed un booleano che indica se deve essere suonata in loop. Si devono fornire metodi per aggiungere in coda una canzone (`addTrack()`), eliminare una canzone dato il titolo (`removeTrack()`) o suonare direttamente una canzone dato il titolo (`playTrack()`). La classe ha un metodo `play()` che suona le canzoni secondo la loro sequenza data.

Si implementi una classe `MusicLibrary` che contiene `Playlist` e consente di inserirle, cercarle e suonarle sulla base del loro nome.

- Nel caso si usi una `std::map<Key K, MappedType T>` si ricorda la presenza dei seguenti metodi:

`T& operator[]( const Key& key );` ritorna un riferimento al valore mappato sulla chiave equivalente a quella passato come argomento, effettuando un inserimento se tale chiave non esiste.

`iterator find( const Key& key );` cerca un elemento con chiave equivalente a quella indicata; rende l'iteratore all'elemento se presente o all'iteratore prima posizione dopo la fine se non presente (`end()`).

`void erase( iterator pos );` rimuove l'elemento alla posizione indicata dall'iteratore

e che i valori presenti nella mappa sono del tipo: `std::pair<const Key, T>` con attributi pubblici `first` per la chiave e `second` per il valore associato.

- Nel caso si usi una `std::list<ValueType T>` si ricorda la presenza dei seguenti metodi:

`void push_back( const T& value );` aggiunge l'elemento in coda alla lista

`iterator erase( iterator pos );` rimuove l'elemento alla posizione indicata dall'iteratore

e l'algoritmo `find`:

```
template< class InputIt, class T >
InputIt find( InputIt first, InputIt last, const T& value );
che cerca nell'intervallo tra l'iteratore first ed il last un valore. L'algoritmo usa
l'operatore == che deve essere definito per il tipo su cui si applica.
```

Si ricorda che la firma di tale operatore è:

```
bool operator==(const T & right) const
```

## 6 (10 punti)

Si consideri la classe astratta `UserLogin` che descrive un'interfaccia per la gestione degli accessi ad un sistema software (es. una banca dati sanitari). La classe `UserOTPLogin` è una classe concreta che estende `UserLogin` e serve a gestire gli accessi in modalità One Time Password (OTP), ovvero inviando un SMS con una password usa e getta ad un utente. Per consentire il login si controlla quindi che nome utente e password temporanea corrispondano con il metodo `checkCredentials()`. Un esempio di uso della classe è riportato nel seguito:

```
UserLogin* userLogin = new UserOTPLogin("Pippo");
// send OTP to user with a SMS
std::string userPassword;
std::string userPhone = "+39 123 456789";
UserOTPLogin* otpLogin = dynamic_cast<UserOTPLogin*>(userLogin);
if (otpLogin)
    userPassword = otpLogin->sendOTP(userPhone);
if (userLogin->checkCredentials("Pippo", userPassword))
    std::cout << "Logging in." << std::endl;
delete userLogin;
```

Le specifiche del sistema richiedono che si consenta anche l'accesso ad utenti che dispongono di una smart card con relativo lettore (es. con tessera sanitaria). Questa funzione è gestita da una vecchia componente software (rappresentata dalla classe `SmartCardReader`) il cui aggiornamento è complesso e costoso.

Per usare questa classe la si deve prima istanziare passando una chiave di criptazione nel costruttore, poi si deve impostare il nome utente e password, con due appositi metodi setter che li memorizzano criptati. Per validare un login si devono passare al metodo `isLoginOK` la versione criptata del nome utente e password, usando il metodo `encryptString()`. Un esempio di uso della classe è riportato nel seguito:

```
SmartCardReader scrLogin("0xDECAF BAD");
scrLogin.setUserName("Tizio");
scrLogin.setPassword("123456");
if (scrLogin.isLoginOK(scrLogin.encryptString("Tizio"),
    scrLogin.encryptString("123456")))
    std::cout << "Logging in." << std::endl;
```

Per semplificare lo sviluppo del sistema software, unificando il codice di gestione degli accessi, si implementi un Adapter che consenta di usare la solita interfaccia di `UserLogin` sia nel caso si usi `UserOTPLogin` sia nel caso l'utente usi ancora il vecchio sistema con Smart Card.

Di seguito sono riportate le classi: `UserLogin`, `UserOTPLogin`.

```
class UserLogin {
public:
    explicit UserLogin(std::string user) : userName(user) {}
    virtual ~UserLogin() {}

    virtual bool checkCredentials(std::string username, std::string password) = 0;
protected:
    std::string userName;
    std::string userPassword;
};

class UserOTPLogin : public UserLogin {
public:
    explicit UserOTPLogin(std::string user) : UserLogin(user) {}

    std::string sendOTP(std::string telephoneNumber);

    bool checkCredentials(std::string username, std::string password) override;

private:
    const int maxSeconds = 20; // time out: after 20 seconds the OTP is not valid anymore
    std::chrono::time_point<std::chrono::system_clock> start;
    std::chrono::time_point<std::chrono::system_clock> end;
    std::string otpPassword;

    std::string generateRandomPassword() {
        userPassword = std::string("password");
        return userPassword;
    }
};

std::string UserOTPLogin::sendOTP(std::string telephoneNumber) {
    start = std::chrono::system_clock::now();
    std::string password = generateRandomPassword();
    std::cout << "Sending password: " << password << " to telephone number: " <<
    telephoneNumber << std::endl;
    return password;
}

bool UserOTPLogin::checkCredentials(std::string userName, std::string password) {
    end = std::chrono::system_clock::now();
    std::chrono::duration<double> diff = end - start;
    if (diff.count() > maxSeconds) {
        std::cerr << "OTP token expired. Please retry." << std::endl;
        return false;
    }
    if ((this->userName == userName) && (this->userPassword == password))
        return true;
    else {
        std::cerr << "Wrong user name or password." << std::endl;
        return false;
    }
}
```

```
class SmartCardReader {
public:
    SmartCardReader(std::string key) : secretKey(key), userPassword(""), userName("") {}

    bool isLoginOK(std::string encryptedUser, std::string encryptedPassword) const {
        if ((userPassword != "") && (userName != "")) {
            if ((encryptedUser == userName) && (encryptedPassword == userPassword))
                return true;
            else
                return false;
        } else
            return false;
    }

    void setUserName(std::string user) {
        userName = encryptString(user);
    }

    void setUserPassword(const std::string &password) {
        this->userPassword = encryptString(password);
    }

    std::string encryptString(std::string value) {
        // compute encrypted version of value string using secretKey
        // return encrypted string...
        return "encrypted_" + secretKey + "_" + value;
    }

private:
    std::string secretKey;
    std::string userPassword;
    std::string userName;
};
```