Esercizio Ridefinizioni Standard

```
class Base {
protected:
    int* data;
    int size;
public:
    virtual ~Base() {}
};

class Derived: public Base {
private:
    double* values;
    char* name;
public:
    // Ridefinizione del costruttore di copia profonda di Derived
    // Ridefinizione dell'operatore di assegnazione profonda di Derived
    // Ridefinizione del distruttore profondo di Derived
};
```

Si considerino le definizioni sopra. Ridefinire le seguenti operazioni standard per la classe `Derived`:

1. Costruttore di copia profonda di `Derived`
2. Operatore di assegnazione profonda di `Derived`
3. Distruttore profondo di `Derived`

Soluzione Esercizio 1

```
// Costruttore di copia profonda
Derived::Derived(const Derived& d): Base(d) {
    size = d.size;
    if(d.values) {
        values = new double[size];
        for(int i = 0; i < size; ++i) {
            values[i] = d.values[i];
        }
    } else {
        values = nullptr;
    }

    if(d.name) {
        int len = strlen(d.name);
        name = new char[len + 1];
        strcpy(name, d.name);
```

```
    } else {
        name = nullptr;
    }

    if(d.data) {
        data = new int[size];
        for(int i = 0; i < size; ++i) {
            data[i] = d.data[i];
        }
    } else {
        data = nullptr;
    }
}

// Operatore di assegnazione profonda
Derived& Derived::operator=(const Derived& d) {
    if(this != &d) {
        // Libera memoria esistente
        delete[] values;
        delete[] name;
        delete[] data;

        // Copia dalla superclasse
        Base::operator=(d);

        // Copia membri locali
        size = d.size;
        if(d.values) {
            values = new double[size];
            for(int i = 0; i < size; ++i) {
                values[i] = d.values[i];
            }
        } else {
            values = nullptr;
        }

        if(d.name) {
            int len = strlen(d.name);
            name = new char[len + 1];
            strcpy(name, d.name);
        } else {
            name = nullptr;
        }

        if(d.data) {
            data = new int[size];
            for(int i = 0; i < size; ++i) {
                data[i] = d.data[i];
            }
        } else {
```

```
            data = nullptr;
        }
    }
    return *this;
}

// Distruttore profondo
Derived::~Derived() {
    delete[] values;
    delete[] name;
    delete[] data;
}
```

Esercizio Modellazione

Esercizio: Sistema di gestione giochi da tavolo

Si consideri il seguente modello di realtà concernente un sistema di gestione per giochi da tavolo.

(A) Definire la seguente gerarchia di classi:

1. Definire una classe base polimorfa astratta `GiocoDaTavolo` i cui oggetti rappresentano un generico gioco da tavolo. Ogni `GiocoDaTavolo` è caratterizzato da un nome, un numero minimo e massimo di giocatori, e una durata media in minuti. La classe è astratta in quanto prevede i seguenti metodi virtuali puri:
   - Un metodo di "clonazione": `GiocoDaTavolo* clone()`.
   - Un metodo `double complessità()` con il seguente contratto puro: `g->complessità()` ritorna un valore da 1.0 a 10.0 che rappresenta la complessità del gioco.
2. Definire una classe concreta `GiocoGestionale` derivata da `GiocoDaTavolo` i cui oggetti rappresentano giochi di gestione risorse. Ogni oggetto `GiocoGestionale` è caratterizzato dal numero di tipi di risorse da gestire. La classe `GiocoGestionale` implementa i metodi virtuali puri di `GiocoDaTavolo` come segue:
   - Implementazione della clonazione standard per la classe `GiocoGestionale`.
   - Per ogni puntatore `p` a `GiocoGestionale`, `p->complessità()` ritorna un valore calcolato come `3.0 + 0.5 * R`, dove `R` è il numero di tipi di risorse.
3. Definire una classe concreta `GiocoDiCarte` derivata da `GiocoDaTavolo` i cui oggetti rappresentano giochi che utilizzano principalmente carte. Ogni oggetto `GiocoDiCarte` è caratterizzato dal numero di carte nel mazzo e da un flag che indica se è un gioco di ruolo. La classe `GiocoDiCarte` implementa i metodi virtuali puri di `GiocoDaTavolo` come segue:
   - Implementazione della clonazione standard per la classe `GiocoDiCarte`.
   - Per ogni puntatore `p` a `GiocoDiCarte`, `p->complessità()` ritorna un valore calcolato come `2.0 + 0.01 * C`, dove `C` è il numero di carte; se il gioco è di ruolo,

il valore viene aumentato del 50%.

4. Definire una classe concreta `GiocoStrategia` derivata da `GiocoDaTavolo` i cui oggetti rappresentano giochi di strategia. Ogni oggetto `GiocoStrategia` è caratterizzato dal numero di fasi di gioco e dalla presenza o meno di un sistema di combattimento. La classe `GiocoStrategia` implementa i metodi virtuali puri di `GiocoDaTavolo` come segue:
   - Implementazione della clonazione standard per la classe `GiocoStrategia`.
   - Per ogni puntatore `p` a `GiocoStrategia`, `p->complessità()` ritorna un valore calcolato come `5.0 + 0.8 * F`, dove `F` è il numero di fasi di gioco; se è presente un sistema di combattimento, il valore viene aumentato di 2.0.

(B) Definire una classe `LudoTeca` i cui oggetti rappresentano una collezione di giochi da tavolo. Un oggetto di `LudoTeca` è caratterizzato da un contenitore di elementi di tipo `const GiocoDaTavolo*` che contiene tutti i giochi della collezione. La classe `LudoTeca` rende disponibili i seguenti metodi:

1. Un metodo `vector<GiocoDiCarte*> filtraPerCarte(int min, int max)` con il seguente comportamento: una invocazione `lt.filtraPerCarte(min, max)` ritorna un vector di puntatori a copie di tutti i giochi di carte nella ludoteca che hanno un numero di carte compreso tra `min` e `max` (inclusi).
2. Un metodo `GiocoDaTavolo* piùComplesso()` con il seguente comportamento: una invocazione `lt.piùComplesso()` ritorna un puntatore ad una copia del gioco con complessità maggiore nella ludoteca; se la ludoteca è vuota, viene sollevata un'eccezione "LudotecaVuota" di tipo `std::string`.
3. Un metodo `void rimuoviComplessi(double soglia)` con il seguente comportamento: una invocazione `lt.rimuoviComplessi(soglia)` rimuove dalla ludoteca tutti i giochi che hanno una complessità maggiore di `soglia`.
4. Un metodo `double complessitàMedia(int giocatori)` con il seguente comportamento: una invocazione `lt.complessitàMedia(giocatori)` ritorna la complessità media di tutti i giochi nella ludoteca che possono essere giocati con esattamente `giocatori` partecipanti; se non ci sono giochi adatti, viene ritornato 0.

Soluzione Esercizio

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

class GiocoDaTavolo {
protected:
    std::string nome;
    int minGiocatori;
    int maxGiocatori;
```

```
    int durata;  // in minuti

public:
    GiocoDaTavolo(const std::string& n, int min, int max, int d):
        nome(n), minGiocatori(min), maxGiocatori(max), durata(d) {}

    virtual ~GiocoDaTavolo() {}

    // Metodi virtuali puri
    virtual GiocoDaTavolo* clone() const = 0;
    virtual double complessità() const = 0;

    // Getters
    std::string getNome() const { return nome; }
    int getMinGiocatori() const { return minGiocatori; }
    int getMaxGiocatori() const { return maxGiocatori; }
    int getDurata() const { return durata; }

    // Metodo per verificare se può essere giocato con un certo numero di
giocatori
    bool puòEssereGiocato(int giocatori) const {
        return giocatori >= minGiocatori && giocatori <= maxGiocatori;
    }
};

class GiocoGestionale : public GiocoDaTavolo {
private:
    int numRisorse;

public:
    GiocoGestionale(const std::string& n, int min, int max, int d, int r):
        GiocoDaTavolo(n, min, max, d), numRisorse(r) {}

    GiocoDaTavolo* clone() const override {
        return new GiocoGestionale(*this);
    }

    double complessità() const override {
        return 3.0 + 0.5 * numRisorse;
    }

    int getNumRisorse() const { return numRisorse; }
};

class GiocoDiCarte : public GiocoDaTavolo {
private:
    int numCarte;
    bool isGiocoRuolo;

public:
```

```
    GiocoDiCarte(const std::string& n, int min, int max, int d, int c, bool
ruolo):
        GiocoDaTavolo(n, min, max, d), numCarte(c), isGiocoRuolo(ruolo) {}

    GiocoDaTavolo* clone() const override {
        return new GiocoDiCarte(*this);
    }

    double complessità() const override {
        double comp = 2.0 + 0.01 * numCarte;
        if (isGiocoRuolo) {
            comp *= 1.5; // Aumento del 50%
        }
        return comp;
    }

    int getNumCarte() const { return numCarte; }
    bool isRuolo() const { return isGiocoRuolo; }
};

class GiocoStrategia : public GiocoDaTavolo {
private:
    int numFasi;
    bool hasCombattimento;

public:
    GiocoStrategia(const std::string& n, int min, int max, int d, int f,
bool combat):
        GiocoDaTavolo(n, min, max, d), numFasi(f), hasCombattimento(combat)
{}

    GiocoDaTavolo* clone() const override {
        return new GiocoStrategia(*this);
    }

    double complessità() const override {
        double comp = 5.0 + 0.8 * numFasi;
        if (hasCombattimento) {
            comp += 2.0;
        }
        return comp;
    }

    int getNumFasi() const { return numFasi; }
    bool hasSistemaCombattimento() const { return hasCombattimento; }
};

class LudoTeca {
private:
    std::vector<const GiocoDaTavolo*> giochi;
```

```
public:
    LudoTeca() {}

    ~LudoTeca() {
        for (auto gioco : giochi) {
            delete gioco;
        }
    }

    // Metodo per aggiungere un gioco alla ludoteca
    void aggiungiGioco(const GiocoDaTavolo& g) {
        giochi.push_back(g.clone());
    }

    // Filtra i giochi di carte con un numero di carte compreso tra min e
max
    std::vector<GiocoDiCarte*> filtraPerCarte(int min, int max) {
        std::vector<GiocoDiCarte*> risultato;

        for (auto gioco : giochi) {
            const GiocoDiCarte* cartaGioco = dynamic_cast<const
GiocoDiCarte*>(gioco);
            if (cartaGioco && cartaGioco->getNumCarte() >= min &&
cartaGioco->getNumCarte() <= max) {
                risultato.push_back(static_cast<GiocoDiCarte*>(cartaGioco-
>clone()));
            }
        }

        return risultato;
    }

    // Trova il gioco più complesso
    GiocoDaTavolo* piùComplesso() {
        if (giochi.empty()) {
            throw std::string("LudotecaVuota");
        }

        auto maxComplessitàIt = std::max_element(giochi.begin(),
giochi.end(),
                                                 [](const GiocoDaTavolo* a,
const GiocoDaTavolo* b) {
                                                     return a->complessità() <
b->complessità();
                                                 });

        return (*maxComplessitàIt)->clone();
    }
```

```
    // Rimuove i giochi con complessità maggiore di una soglia
    void rimuoviComplessi(double soglia) {
        auto newEnd = std::remove_if(giochi.begin(), giochi.end(),
                                     [soglia](const GiocoDaTavolo* g) {
                                         bool rimuovi = g->complessità() >
soglia;
                                         if (rimuovi) {
                                             delete g;
                                         }
                                         return rimuovi;
                                     });

        giochi.erase(newEnd, giochi.end());
    }

    // Calcola la complessità media dei giochi adatti per un certo numero di
giocatori
    double complessitàMedia(int giocatori) {
        int count = 0;
        double somma = 0.0;

        for (auto gioco : giochi) {
            if (gioco->puòEssereGiocato(giocatori)) {
                somma += gioco->complessità();
                count++;
            }
        }

        return count > 0 ? somma / count : 0.0;
    }
};
```

Esercizi Funzione

Esercizio 1 - Funzione di gestione documenti

Si assumano le seguenti specifiche riguardanti una libreria di gestione documenti.

(a) `Document` è la classe base polimorfa di tutti i documenti. La classe `Document` rende disponibile un metodo `int getSize() const` con il seguente comportamento: `doc.getSize()` ritorna la dimensione in KB del documento `doc`. Inoltre, la classe `Document` rende disponibile un metodo `bool isReadOnly() const` con il seguente comportamento: `doc.isReadOnly()` ritorna `true` se il documento `doc` è in sola lettura, altrimenti ritorna `false`.

(b) `TextDocument` è derivata direttamente da `Document` ed è la classe dei documenti testuali. La classe `TextDocument` rende disponibile un metodo `std::string getFormat() const` con il seguente comportamento: `text.getFormat()` ritorna il formato del documento

testuale `text` (ad esempio "txt", "md", "docx"). Inoltre, `TextDocument` rende disponibile un metodo `void encrypt()` con il seguente comportamento: `text.encrypt()` cripta il documento testuale `text`.

(c) `ImageDocument` è derivata direttamente da `Document` ed è la classe dei documenti immagine. La classe `ImageDocument` rende disponibile un metodo `int getResolution() const` con il seguente comportamento: `img.getResolution()` ritorna la risoluzione in DPI dell'immagine `img`. Inoltre, `ImageDocument` rende disponibile un metodo `void resize(double factor)` con il seguente comportamento: `img.resize(factor)` ridimensiona l'immagine `img` di un fattore `factor`.

Definire una funzione `list<Document*> processDocs(vector<Document*>&, int)` tale che in ogni invocazione `processDocs(docs, limit)`:

(1) Per ogni puntatore `p` contenuto nel vector `docs`:

- Se `p` punta ad un oggetto che è un `TextDocument` con formato "docx" e non è in sola lettura, viene criptato.
- Se `p` punta ad un oggetto che è un `ImageDocument` con risoluzione maggiore di 300 DPI e dimensione maggiore di `limit` KB, viene ridimensionato di un fattore 0.5.
- Quindi, se l'oggetto risultante ha una dimensione maggiore di `limit` KB, il puntatore viene rimosso dal vector `docs`.

(2) L'invocazione `processDocs(docs, limit)` deve ritornare una list contenente puntatori a copie di tutti i documenti che sono stati rimossi dal vector `docs`.

Esercizio 2 - Funzione per la gestione di componenti UI

Si assumano le seguenti specifiche riguardanti una libreria per interfacce utente.

(a) `UIComponent` è la classe base polimorfa di tutti i componenti dell'interfaccia utente. La classe `UIComponent` rende disponibile un metodo `bool isVisible() const` con il seguente comportamento: `comp.isVisible()` ritorna `true` se il componente `comp` è visibile, altrimenti ritorna `false`. Inoltre, la classe `UIComponent` rende disponibile un metodo `void setVisible(bool v)` con il seguente comportamento: `comp.setVisible(v)` imposta la visibilità del componente `comp` al valore `v`.

(b) `InteractiveComponent` è derivata direttamente da `UIComponent` ed è la classe base dei componenti interattivi. La classe `InteractiveComponent` rende disponibile un metodo `bool isEnabled() const` con il seguente comportamento: `ic.isEnabled()` ritorna `true` se il componente interattivo `ic` è abilitato, altrimenti ritorna `false`. Inoltre, `InteractiveComponent` rende disponibile un metodo `void setEnabled(bool e)` con il seguente comportamento: `ic.setEnabled(e)` imposta lo stato di abilitazione del componente interattivo `ic` al valore `e`.

(c) `Button` è derivata direttamente da `InteractiveComponent` ed è la classe dei pulsanti. La classe `Button` rende disponibile un metodo `std::string getLabel() const` con il seguente comportamento: `btn.getLabel()` ritorna l'etichetta del pulsante `btn`. Inoltre, `Button` rende disponibile un metodo `void setLabel(const std::string& label)` con il seguente comportamento: `btn.setLabel(label)` imposta l'etichetta del pulsante `btn` al valore `label`.

(d) `TextInput` è derivata direttamente da `InteractiveComponent` ed è la classe dei campi di input testuale. La classe `TextInput` rende disponibile un metodo `bool isReadOnly() const` con il seguente comportamento: `input.isReadOnly()` ritorna `true` se il campo di input `input` è in sola lettura, altrimenti ritorna `false`. Inoltre, `TextInput` rende disponibile un metodo `void setReadOnly(bool ro)` con il seguente comportamento: `input.setReadOnly(ro)` imposta lo stato di sola lettura del campo di input `input` al valore `ro`.

Definire una funzione `std::pair<vector<UIComponent*>, vector<Button*>> processUI(const list<UIComponent*>&, const std::string&)` tale che in ogni invocazione `processUI(components, prefix)`:

(1) Per ogni puntatore `p` contenuto nella lista `components`:

- Se `p` punta ad un oggetto che è un `Button` visibile, modifica l'etichetta del `Button` aggiungendo `prefix` all'inizio dell'etichetta corrente.
- Se `p` punta ad un oggetto che è un `TextInput` visibile e non è in sola lettura, lo imposta come in sola lettura.
- Se `p` punta ad un oggetto che è un `InteractiveComponent` non visibile, lo rende visibile ma disabilitato.

(2) L'invocazione `processUI(components, prefix)` deve ritornare una coppia composta da:

- Un vector contenente puntatori a copie di tutti gli `UIComponent` che erano visibili prima dell'applicazione delle modifiche del punto (1).
- Un vector contenente puntatori a copie di tutti i `Button` le cui etichette sono state modificate.

Esercizio 3 - Funzione per la gestione di dispositivi di rete

Si assumano le seguenti specifiche riguardanti una libreria di gestione di dispositivi di rete.

(a) `NetworkDevice` è la classe base polimorfa di tutti i dispositivi di rete. La classe `NetworkDevice` rende disponibile un metodo `std::string getIP() const` con il seguente comportamento: `dev.getIP()` ritorna l'indirizzo IP del dispositivo `dev`. Inoltre, la classe `NetworkDevice` rende disponibile un metodo `bool isConnected() const` con il seguente

comportamento: `dev.isConnected()` ritorna `true` se il dispositivo `dev` è connesso alla rete, altrimenti ritorna `false`.

(b) `RouterDevice` è derivata direttamente da `NetworkDevice` ed è la classe dei router. La classe `RouterDevice` rende disponibile un metodo `int getActivePorts() const` con il seguente comportamento: `router.getActivePorts()` ritorna il numero di porte attivamente in uso nel router `router`. Inoltre, `RouterDevice` rende disponibile un metodo `void restartRouter()` con il seguente comportamento: `router.restartRouter()` riavvia il router `router`.

(c) `FirewallDevice` è derivata direttamente da `NetworkDevice` ed è la classe dei firewall. La classe `FirewallDevice` rende disponibile un metodo `bool hasSecurityAlert() const` con il seguente comportamento: `fw.hasSecurityAlert()` ritorna `true` se il firewall `fw` ha rilevato un allarme di sicurezza, altrimenti ritorna `false`. Inoltre, `FirewallDevice` rende disponibile un metodo `void clearAlerts()` con il seguente comportamento: `fw.clearAlerts()` cancella tutti gli allarmi nel firewall `fw`.

(d) `WirelessRouter` è derivata direttamente da `RouterDevice` ed è la classe dei router wireless. La classe `WirelessRouter` rende disponibile un metodo `std::string getSSID() const` con il seguente comportamento: `wr.getSSID()` ritorna l'SSID della rete wireless gestita dal router wireless `wr`. Inoltre, `WirelessRouter` rende disponibile un metodo `void changeChannel(int channel)` con il seguente comportamento: `wr.changeChannel(channel)` cambia il canale di trasmissione del router wireless `wr` al valore `channel`.

Definire una funzione `std::map<std::string, NetworkDevice*> manageNetwork(vector<NetworkDevice*>&, bool)` tale che in ogni invocazione `manageNetwork(devices, fullReset)`:

(1) Per ogni puntatore `p` contenuto nel vector `devices`:

- Se `p` punta ad un oggetto che è un `RouterDevice` con più di 10 porte attive, riavvia il router.
- Se `p` punta ad un oggetto che è un `FirewallDevice` con allarmi di sicurezza, cancella gli allarmi.
- Se `p` punta ad un oggetto che è un `WirelessRouter` con SSID che inizia con "Guest_" e il parametro `fullReset` è `true`, riavvia il router e cambia il canale a 6.
- Se `p` punta ad un oggetto che non è connesso alla rete, lo rimuove dal vector `devices`.

(2) L'invocazione `manageNetwork(devices, fullReset)` deve ritornare una mappa che associa gli indirizzi IP a puntatori a copie di tutti i dispositivi di rete che sono stati rimossi dal vector `devices`.

Soluzioni degli Esercizi

Soluzione Esercizio 1

```
#include <iostream>
#include <vector>
#include <list>
#include <string>

class Document {
public:
    virtual int getSize() const = 0;
    virtual bool isReadOnly() const = 0;
    virtual ~Document() {}
    virtual Document* clone() const = 0;
};

class TextDocument : public Document {
public:
    std::string getFormat() const { /* implementazione */ }
    void encrypt() { /* implementazione */ }
    virtual TextDocument* clone() const override { /* implementazione */ }
};

class ImageDocument : public Document {
public:
    int getResolution() const { /* implementazione */ }
    void resize(double factor) { /* implementazione */ }
    virtual ImageDocument* clone() const override { /* implementazione */ }
};

list<Document*> processDocs(vector<Document*>& docs, int limit) {
    list<Document*> removedDocs;
    auto it = docs.begin();

    while (it != docs.end()) {
        Document* doc = *it;

        // Controlla se è un TextDocument docx non in sola lettura
        TextDocument* textDoc = dynamic_cast<TextDocument*>(doc);
        if (textDoc && textDoc->getFormat() == "docx" && !textDoc-
>isReadOnly()) {
            textDoc->encrypt();
        }

        // Controlla se è un ImageDocument con risoluzione > 300 DPI e
dimensione > limit
        ImageDocument* imgDoc = dynamic_cast<ImageDocument*>(doc);
        if (imgDoc && imgDoc->getResolution() > 300 && imgDoc->getSize() >
limit) {
            imgDoc->resize(0.5);
```

```
        }

        // Verifica se la dimensione risultante è > limit
        if (doc->getSize() > limit) {
            removedDocs.push_back(doc->clone());
            it = docs.erase(it);
        } else {
            ++it;
        }
    }

    return removedDocs;
}
```

Soluzione Esercizio 2

```
#include <iostream>
#include <vector>
#include <list>
#include <string>
#include <utility>

class UIComponent {
public:
    virtual bool isVisible() const = 0;
    virtual void setVisible(bool v) = 0;
    virtual ~UIComponent() {}
    virtual UIComponent* clone() const = 0;
};

class InteractiveComponent : public UIComponent {
public:
    virtual bool isEnabled() const = 0;
    virtual void setEnabled(bool e) = 0;
    virtual InteractiveComponent* clone() const override = 0;
};

class Button : public InteractiveComponent {
public:
    virtual std::string getLabel() const = 0;
    virtual void setLabel(const std::string& label) = 0;
    virtual Button* clone() const override = 0;
};

class TextInput : public InteractiveComponent {
public:
    virtual bool isReadOnly() const = 0;
    virtual void setReadOnly(bool ro) = 0;
```

```
    virtual TextInput* clone() const override = 0;
};

std::pair<vector<UIComponent*>, vector<Button*>> processUI(const
list<UIComponent*>& components, const std::string& prefix) {
    vector<UIComponent*> originalVisibleComponents;
    vector<Button*> modifiedButtons;

    for (auto p : components) {
        // Salva i componenti originalmente visibili
        if (p->isVisible()) {
            originalVisibleComponents.push_back(p->clone());
        }

        // Processa i Button visibili
        Button* btn = dynamic_cast<Button*>(p);
        if (btn && btn->isVisible()) {
            std::string originalLabel = btn->getLabel();
            btn->setLabel(prefix + originalLabel);
            modifiedButtons.push_back(static_cast<Button*>(btn->clone()));
        }

        // Processa i TextInput visibili e non in sola lettura
        TextInput* txtInput = dynamic_cast<TextInput*>(p);
        if (txtInput && txtInput->isVisible() && !txtInput->isReadOnly()) {
            txtInput->setReadOnly(true);
        }

        // Processa gli InteractiveComponent non visibili
        InteractiveComponent* ic = dynamic_cast<InteractiveComponent*>(p);
        if (ic && !ic->isVisible()) {
            ic->setVisible(true);
            ic->setEnabled(false);
        }
    }

    return {originalVisibleComponents, modifiedButtons};
}
```

Soluzione Esercizio 3

```
#include <iostream>
#include <vector>
#include <map>
#include <string>

class NetworkDevice {
public:
```

```
    virtual std::string getIP() const = 0;
    virtual bool isConnected() const = 0;
    virtual ~NetworkDevice() {}
    virtual NetworkDevice* clone() const = 0;
};

class RouterDevice : public NetworkDevice {
public:
    virtual int getActivePorts() const = 0;
    virtual void restartRouter() = 0;
    virtual RouterDevice* clone() const override = 0;
};

class FirewallDevice : public NetworkDevice {
public:
    virtual bool hasSecurityAlert() const = 0;
    virtual void clearAlerts() = 0;
    virtual FirewallDevice* clone() const override = 0;
};

class WirelessRouter : public RouterDevice {
public:
    virtual std::string getSSID() const = 0;
    virtual void changeChannel(int channel) = 0;
    virtual WirelessRouter* clone() const override = 0;
};

std::map<std::string, NetworkDevice*> manageNetwork(vector<NetworkDevice*>&
devices, bool fullReset) {
    std::map<std::string, NetworkDevice*> removedDevices;
    auto it = devices.begin();

    while (it != devices.end()) {
        NetworkDevice* device = *it;

        // Gestisci RouterDevice con più di 10 porte attive
        RouterDevice* router = dynamic_cast<RouterDevice*>(device);
        if (router && router->getActivePorts() > 10) {
            router->restartRouter();
        }

        // Gestisci FirewallDevice con allarmi di sicurezza
        FirewallDevice* firewall = dynamic_cast<FirewallDevice*>(device);
        if (firewall && firewall->hasSecurityAlert()) {
            firewall->clearAlerts();
        }

        // Gestisci WirelessRouter con SSID che inizia con "Guest_" e
fullReset è true
        WirelessRouter* wirelessRouter = dynamic_cast<WirelessRouter*>
```

```
(device);
        if (wirelessRouter && wirelessRouter->getSSID().substr(0, 6) ==
"Guest_" && fullReset) {
            wirelessRouter->restartRouter();
            wirelessRouter->changeChannel(6);
        }

        // Rimuovi dispositivi non connessi
        if (!device->isConnected()) {
            removedDevices[device->getIP()] = device->clone();
            it = devices.erase(it);
        } else {
            ++it;
        }
    }

    return removedDevices;
}
```

Esercizio Lambda

```
template<class T>
class SmartPtr {
private:
    T* ptr;
public:
    SmartPtr(T* p = nullptr): ptr(p) { cout << "SmartPtr() "; }
    SmartPtr(const SmartPtr& s): ptr(s.ptr) { cout << "SmartPtrCopy() "; }
    ~SmartPtr() { cout << "~SmartPtr() "; }
    T& operator*() { return *ptr; }
    T* operator->() { return ptr; }
};

template<class T>
vector<int> analizza(const vector<T>& v, function<bool(T)> pred) {
    vector<int> r;
    for(int i = 0; i < v.size(); ++i)
        if(pred(v[i])) r.push_back(i);
    return r;
}

template<class T>
int conta(const vector<T>& v, T val) {
    auto risultato = analizza(v, [val](T x) { return x > val; });
    return risultato.size();
}

int somma(const vector<int>& v) {
```

```
    int s = 0;
    for(auto x : v) s += x;
    return s;
}

vector<int> v1 = {5, -2, 8, 3, 5, 2, 7};
vector<double> v2 = {2.5, 4.3, -1.2, 0.8, 6.7};
```

Queste definizioni compilano correttamente (con opportuni `#include` e using). Per ognuno dei seguenti statement scrivere nell'apposito spazio:

- NON COMPILA se la compilazione dello statement provoca un errore;
- UNDEFINED se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se lo statement compila ed esegue correttamente (senza undefined behaviour o errori run-time) allora si scriva la stampa che l'esecuzione produce in output su cout; se non provoca alcuna stampa allora si scriva NESSUNA STAMPA.

```
cout << conta(v1, 3);
.....................................................................
.......
cout << conta(v2, 1.5);
.....................................................................
.....
cout << somma(analizza(v1, [](int x){ return x % 2 == 0; }));
.....................................
SmartPtr<int> p1(new int(5)); cout << *p1;
.........................................................
SmartPtr<int> p2 = p1; cout << *p2;
.............................................................
```

Soluzione:

```
cout << conta(v1, 3);
.....................................................................
..... 4
cout << conta(v2, 1.5);
.....................................................................
... 3
cout << somma(analizza(v1, [](int x){ return x % 2 == 0; }));
..................................... 3
SmartPtr<int> p1(new int(5)); cout << *p1;
..................................................... SmartPtr() 5
```

```
SmartPtr<int> p2 = p1; cout << *p2;
.............................................................
SmartPtrCopy() 5
```

Esercizio Analisi di Compilazione con Template

```
#include<iostream>
#include<string>
using namespace std;

class Numero {
public:
    operator int() const { return 42; }
};

template<class T> class Parser; // dichiarazione incompleta

template<class T1, class T2 = int, int K = 0>
class Dato {
    friend class Parser<T1>;
private:
    T1 val1;
    T2 val2;
    int count;
public:
    Dato(int c = K): count(c) {}
};

template<class T>
class Parser {
public:
    void f() const { Dato<T> d; cout << d.val1 << endl; }
    void g() const { Dato<int, T> d; cout << d.val2 << endl; }
    void h() const { Dato<T, double, 3> d; cout << d.count << endl; }
    void m() const { Dato<char, T> d; cout << d.val2 << endl; }
    void n() const { Dato<string, T, 5> d; cout << d.count << endl; }
    void o() const { Dato<Numero, T, 7> d(10); cout << d.count << endl; }
};
```

Determinare se i seguenti main() compilano correttamente o meno barrando la corrispondente scritta.

```
int main() { Parser<int> p1; p1.f(); }              COMPILA □ NON COMPILA □
int main() { Parser<string> p2; p2.f(); }           COMPILA □ NON COMPILA □
int main() { Parser<double> p3; p3.g(); }           COMPILA □ NON COMPILA □
int main() { Parser<char> p4; p4.h(); }             COMPILA □ NON COMPILA □
```

```
int main() { Parser<double> p5; p5.h(); }          COMPILA □ NON COMPILA □
int main() { Parser<int> p6; p6.m(); }             COMPILA □ NON COMPILA □
int main() { Parser<char> p7; p7.n(); }            COMPILA □ NON COMPILA □
int main() { Parser<double> p8; p8.n(); }          COMPILA □ NON COMPILA □
int main() { Parser<int> p9; p9.o(); }             COMPILA □ NON COMPILA □
int main() { Parser<Numero> p10; p10.o(); }        COMPILA □ NON COMPILA □
```