Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 25/1/2019

Nome..... Cognome..... Matricola.....

Esercizio 1

Si consideri il seguente modello concernente gli specialisti ICT (Information and Communication Technology) che lavorano per la internet company Amazonia©.

(A) Definire la seguente gerarchia di classi.

1. Definire una classe base polimorfa astratta `ICTstaff` i cui oggetti rappresentano uno specialista ICT che lavora in Amazonia. Ogni `ICTStaff` è caratterizzato da uno stipendio fisso mensile contrattato individualmente, dal titolo di laurea posseduto e dal possedere una laurea triennale o magistrale, dove il titolo di laurea è un valore del seguente tipo enumerativo: `enum Laurea {Informatica, Ingegneria, Altro};` La classe è astratta in quanto prevede i seguenti metodi virtuali puri:
   - un metodo di "clonazione" con l'usuale contratto di "costruzione di copia polimorfa".
   - un metodo `double salary()` con il seguente contratto: `p->salary()` ritorna lo stipendio mensile per il mese corrente di `*p`.
2. Definire una classe concreta `SwEngineer` derivata da `ICTStaff` i cui oggetti rappresentano un ingegnere software che lavora in Amazonia. Ogni `SwEngineer` è caratterizzato dall'occuparsi di sicurezza oppure no. La classe `SwEngineer` implementa i metodi virtuali puri di `ICTStaff` come segue:
   - implementazione del metodo di clonazione specifica del tipo `SwEngineer`;
   - per ogni puntatore `p` a `SwEngineer`, l'invocazione `p->salary()` ritorna lo stipendio fisso mensile di `*p`, aumentato di 500 € se `*p` si occupa di sicurezza software.
3. Definire una classe concreta `HwEngineer` derivata da `ICTStaff` i cui oggetti rappresentano un ingegnere hardware che lavora in Amazonia. Ogni `HwEngineer` è caratterizzato dal numero di trasferte di lavoro effettuate nel mese corrente. La classe `HwEngineer` implementa i metodi virtuali puri di `ICTStaff` come segue:
   - implementazione del metodo di clonazione specifica del tipo `HwEngineer`;
   - per ogni puntatore `p` a `HwEngineer`, l'invocazione `p->salary()` ritorna lo stipendio fisso mensile di `*p`, aumentato di 300 € per ogni trasferta di lavoro effettuata da `*p` nel mese corrente.

(B) Definire una classe `Amazonia` i cui oggetti rappresentano gli specialisti ICT che correntemente lavorano in Amazonia. La classe `Amazonia` deve soddisfare le seguenti specifiche:

4. È definita una classe annidata `SmartP` i cui oggetti rappresentano un puntatore polimorfo smart a `ICTStaff`.
   - La classe annidata `SmartP` deve essere dotata di un costruttore `SmartP(ICTstaff*)` con il seguente comportamento: `SmartP(q)` costruisce un oggetto `SmartP` il cui puntatore polimorfo punta ad una copia dell'oggetto `*q`.
   - La classe `SmartP` ridefinisce costruttore di copia profonda, assegnazione profonda e distruttore profondo.
   - La classe `SmartP` ridefinisce gli operatori di dereferenziazione `*` e di accesso mediante freccia `->` nell'usuale modo che permetta di usare i puntatori smart di `SmartP` come puntatori ordinari.
5. Un oggetto di `Amazonia` è caratterizzato da un contenitore di oggetti di tipo `SmartP` che memorizza i puntatori smart a tutti e soli gli specialisti ICT che lavorano in Amazonia.
6. La classe `Amazonia` rende disponibili i seguenti metodi:
   - 6.1 Un metodo `bool insert(SwEngineer*, unsigned int)` con il seguente comportamento: una invocazione `am.insert(p,k)` inserisce un puntatore smart ad una copia di `*p` nel contenitore di `am` se il numero di ingegneri software di Amazonia che si occupano di sicurezza è minore di `k`, altrimenti non viene effettuato l'inserimento; se l'inserimento viene effettuato allora si ritorna `true`, altrimenti `false`.
   - 6.2 Un metodo `vector<HwEngineer> fire(double)` con il seguente comportamento: una invocazione `am.fire(s)` elimina dal contenitore di `am` tutti gli specialisti ICT di Amazonia che hanno uno stipendio mensile maggiore di `s`; inoltre ritorna un vector di `HwEngineer` che contiene tutti e soli gli ingegneri hardware eliminati.
   - 6.3 Un metodo `vector<SwEngineer*> masterInf()` con il seguente comportamento: una invocazione `am.masterInf()` ritorna un vector di puntatori ordinari a `SwEngineer` contenente tutti e soli gli ingegneri software di Amazonia in possesso di una Laurea Magistrale in Informatica.

Esercizio 2

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  virtual void f() {cout <<" B::f "; g(); j();}
  virtual void g() const {cout <<" B::g ";}
  virtual const B* j() {cout<<" B::j "; return this;}
  virtual void k() {cout <<" B::k "; j(); m(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual B& n() {cout <<" B::n "; return *this;}
};
```

```
class C: virtual public B {
public:
  C() {cout<< " C() ";}
  ~C() {cout<< " ~C() ";}
  virtual void g() const override {cout <<" C::g ";}
  void k() override {cout <<" C::k "; B::n();}
  virtual void m() {cout <<" C::m "; g(); j();}
  B& n() override {cout <<" C::n "; return *this;}
};
```

```
class D: virtual public B {
public:
  D() {cout<< " D() ";}
  ~D() {cout<< " ~D() ";}
  virtual void g() {cout <<" D::g ";}
  const B* j() {cout <<" D::j "; return this;}
  void k() const {cout <<" D::k "; k();}
  void m() {cout <<" D::m "; g(); j();}
};
```

```
class E: public C, public D  {
public:
  E() {cout<< " E() ";}
  ~E() {cout<< " ~E() ";}
  virtual void g() const {cout <<" E::g ";}
  const E* j() {cout <<" E::j "; return this;}
  void m() {cout <<" E::m "; g(); j();}
  D& n() final {cout <<" E::n "; return *this;}
};
```

```
class F: public E {
public:
  F() {cout<< " F() ";}
  ~F() {cout<< " ~F() ";}
  F(const F& x): B(x) {cout<< " Fc ";}
  void k() {cout <<" F::k "; g();}
  void m() {cout <<" F::m "; j();}
};
```

```
B* p1 = new E(); B* p2 = new C(); B* p3 = new D(); C* p4 = new E();
const B* p5 = new D(); const B* p6 = new E(); const B* p7 = new F(); F f;
```

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**.

```
F x; ..........

C* p = new F(f); ..........

p1->f(); ..........

p1->m(); ..........

(p1->j())->k(); ..........

(dynamic_cast<const F*>(p1->j()))->g(); ..........

p2->f(); ..........

p2->m(); ..........

(p2->j())->g(); ..........

p3->f(); ..........

p3->k(); ..........

(p3->n()).m(); ..........

(dynamic_cast<D&>(p3->n())).g(); ..........

p4->f(); ..........

p4->k(); ..........

(p4->n()).m(); ..........

(p5->n()).g(); ..........

(dynamic_cast<E*>(p6))->j(); ..........

(dynamic_cast<C*>(const_cast<B*>(p7)))->k(); ..........

delete p7; ..........
```