Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 23/01/23

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

Esercizio Funzione

```
class A {
public:
  virtual A* f() const =0;
};

class B: public A {};

class C: public B {
public:
  B* f() const {return new C();}
};

class D: public B {};

class E: public B {
public:
  A* f() const {return new E();}
};

class F: public C, public D, public E {
public:
  D* f() const {return new F();}
};
```

Queste definizioni compilano correttamente. Definire una funzione

`list<const D *const> fun(const vector<const B*>&)`

con il seguente comportamento: in ogni invocazione `fun(v)`, per tutti i puntatori `q` contenuti nel vector `v`:

(A) se `q` non è nullo ed ha un tipo dinamico esattamente uguale a `C` allora `q` deve essere rimosso da `v`;

(A_1) se il numero N di puntatori rimossi dal vector `v` è maggiore di 2 allora viene sollevata una eccezione di tipo `C`.

(B) sul puntatore `q` non nullo deve essere invocata la funzione virtuale pura `A* A::f()` che ritorna un puntatore che indichiamo qui con `ptr`;

(B_1) se `ptr` è nullo allora viene sollevata una eccezione `std::string("nullptr");`

(B_2) `fun` ritorna la lista di tutti e soli questi puntatori `ptr` che: non sono nulli ed hanno un tipo dinamico che è sottotipo di `D*` e non è un sottotipo di `E*`.

Esercizio Cosa Stampa

```
template<class Functor>
vector<int> find_template(const vector<int>& v, Functor t) {
    vector<int> r;
    for(auto it = v.begin(); it != v.end(); ++it) if (t(*it)) r.push_back(*it);
    return r;
}

unsigned int find_1(const vector<int>& v,int k) {
  vector<int> w = find_template(v, [v,k] (int n) { return n>k; } );
  return w.size();
}

vector<int> find_2(const vector<int>& v) {
  return find_template(v, [v] (int n) { return n<v.size(); } );
}

vector<int> v1 = {3,6,4,6,2,5,-2,4,2}; vector<int> v2 = {-2,-6,4,4,2,5,0,4,2,3,2,0};
```

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 << find_1(v1,2); ..........................................................................

cout << find_1(v2,2); ..........................................................................

cout << find_2(v1).size(); .....................................................................

cout << find_2(v2).size(); .....................................................................
```

Esercizio Cosa Stampa

```
class B {
public:
  B() {cout<< " B() ";}
  virtual ~B() {cout<< " ~B() ";}
  virtual void g() const {cout <<" B::g ";}
  virtual const B* j() {cout<<" B::j "; n(); 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() ";}
  void g() const {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: virtual 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 E(); const B* p6 = new 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**.

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

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

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

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

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

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

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

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

(dynamic_cast<E*>(p5))->j(); ..........

(dynamic_cast<C*>(const_cast<B*>(p6)))->k(); ..........
```