Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 25/01/2024

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

Esercizio Cosa Stampa

```
class A {
public:
  A() {cout<< " A() ";}
  ~A() {cout<< " ~A ";}
  A(const A& x) {cout<< " Ac ";}
  virtual const A* j() {cout<<" A::j "; return this;}
  virtual void k() {cout <<" A::k "; m();}
  void m() {cout <<" A::m "; j();}
};

class B: virtual public A {
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;}
  void k() {cout <<" B::k "; j(); m(); }
  void m() {cout <<" B::m "; g(); j();}
  virtual A& 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 ";}
  E(const E& x) {cout<< " Ec ";}
  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;}
};

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

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **UNDEFINED** se lo statement compila correttamente ma la sua esecuzione provoca un undefined behaviour o un errore run-time;
- se l'istruzione compila correttamente e non provoca errori a 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**.

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

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

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

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

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

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

((dynamic_cast<D*>(p4))->n()).k(); .....

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

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

new E(*p7); .....

delete p1; .....

delete p4; .....
```

Esercizio Tipi. Siano `A`, `B`, `C`, `D` e `E` cinque **diverse** classi polimorfe. Si considerino le seguenti definizioni.

```
template <class X, class Y>
const X* fun(X& r) {
  const X* ptr = &r; return dynamic_cast<Y*>(ptr);
}
int main() {
  B b; C c; D d; E e;
  if( fun<A,B>(c) == nullptr ) cout << "Bjarne ";
  if( dynamic_cast<C*>(&b) == nullptr ) cout << "Stroustrup";
  const A* p = fun<D,B>(d);
  const D* q = fun<E,B>(e);
  fun<C,D>(d);
}
```

Si supponga che:

1. il `main()` compili correttamente ed esegua senza provocare errori a run-time o undefined behaviour;
2. l'esecuzione del `main()` provochi in output su `cout` la stampa `Bjarne Stroustrup`.

In tali ipotesi, per ognuna delle relazioni di sottotipo `T1`$\leq$`T2` nella seguente **tabella da ricopiare nel foglio** scrivere nella corrispondente cella:

(a) "VERO" per indicare che `T1` **sicuramente è** sottotipo di `T2`;

(b) "FALSO" per indicare che `T1` **sicuramente non è** sottotipo di `T2`;

(c) "POSSIBILE" **altrimenti**, ovvero se non valgono nè (a) nè (b).

A$\leq$B	A$\leq$C	A$\leq$D	A$\leq$E	B$\leq$E	D$\leq$E

Esercizio Funzione Definire un template di funzione `template<class T> list<const iostream*> fun(vector<ostream*>&)` con il seguente comportamento: in ogni invocazione `fun(v)`, per ogni puntatore `p` elemento (di tipo `ostream*`) del vector `v`:

1. se `p` non è nullo e `*p` è un `fstream` che non è nello stato good (ovvero stato 0, con tutti i bit di errore spenti), allora `p` diventa nullo;
2. se `p` non è nullo e `*p` è uno `stringstream` nello stato good, allora `p` viene inserito nella lista che la funzione deve ritornare;
3. se la lista che la funzione deve ritornare è vuota allora la funzione solleva una eccezione di tipo `T`, altrimenti ritorna la lista.