Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 20/2/2020

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

Esercizio 1

Definire un template di classe `Vettore<T,sz>` i cui oggetti rappresentano un array di tipo `T` e dimensione `sz` ≥ 0. Il template di classe `Vettore<T,sz>` deve includere: un costruttore `Vettore(const T& x)` che costruisce un array in cui tutte le celle memorizzano il valore `x`; un costruttore di default `Vettore()` che costruisce un array in cui tutte le celle memorizzano il valore di default `T()` del tipo `T`; ridefinizione di costruttore di copia profonda, assegnazione profonda e distruzione profonda; overloading degli operatori `operator*` di dereferenziazione e `operator[]` di indicizzazione con comportamento analogo ai corrispondenti operatori disponibili per gli array ordinari; overloading dell'operatore di output.

Ad esempio, il seguente codice dovrà compilare correttamente e l'esecuzione dovrà **provocare esattamente** le stampe riportate nei commenti.

```
Vettore<int,4> v1(2); Vettore< Vettore<int,3> ,3> v2(3); Vettore<int,4> v3(v1); Vettore<int,4> v4;
v3[2]=6;
*v1=9;
v4=v1;
v4[3]=5
std::cout << v1 << std::endl; // 9 2 2 2
std::cout << v2 << std::endl; // 3 3 3  3 3 3  3 3 3
std::cout << v3 << std::endl; // 2 2 6 2
std::cout << v4 << std::endl; // 9 2 2 5
```

Esercizio 2

Si considerino le seguenti definizioni:

```
class A {
public:
  virtual ~A() {}
};

class B: public A {};

class C: public B {};

class D: public B {};

class E: public C {};

char F(A* pa, B& rb) {
  B* p = dynamic_cast<B*> (pa);
  C* q = dynamic_cast<C*> (pa);
  if(dynamic_cast<E*> (&rb)) {
    if(p || q) return 'M';
    else return 'I';
  }
  if(dynamic_cast<C*> (&rb)) return 'L';
  if(q) return 'A';
  if(p) return 'N';
  return 'O';
}
```

Si consideri inoltre il seguente `main()` incompleto:

```
int main() {
  A a; B b; C c; D d; E e;

  cout << F(&b,e) << F(&a,e) << F(&a,c) << F(&c,b) << F(&b,b) << F(&a,b);
}
```

Definire opportunamente negli appositi spazi con puntini le chiamate alla funzione `F()` in questo `main()` usando gli oggetti locali `a`, `b`, `c`, `d`, `e`, `f` in modo tale che non vi siano errori in compilazione o a run-time e l'esecuzione produca in output esattamente la stampa **`MILANO`**.

Esercizio 3

Si considerino le seguenti definizioni.

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

class Z {
public:
  operator int() const {return 0;}
};

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

template<class T1, class T2 = Z, int k = 1>
class C {
  friend class D<T1>;
private:
  T1 t1;
  T2 t2;
  int a;
  C(int x =k): a(x) {}
};

template<class T>
class D {
 public:
  void f() const {C<T,T> c(1); cout << c.t1 << c.t2 << c.a;}
  void g() const {C<int> c;}
  void h() const {C<T,int> c(3); cout << c.t2 << c.a;}
  void m() const {C<int,T,3> c; cout << c.t1;}
  void n() const {C<int,double> c; cout << c.t1 << c.t2 << c.a;}
  void o() const {C<char,double> c(6); cout << c.a;}
  void p() const {C<Z,T,7> c(7); cout << c.t2 << c.a;}
};
```

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

`int main() { D<char> d1; d1.f(); }`	COMPILA
`int main() { D<std::string> d2; d2.f(); }`	COMPILA
`int main() { D<char> d3; d3.g(); }`	NON COMPILA
`int main() { D<int> d4; d4.g(); }`	COMPILA
`int main() { D<char> d5; d5.h(); }`	COMPILA
`int main() { D<int> d6; d6.h(); }`	COMPILA
`int main() { D<char> d7; d7.m(); }`	NON COMPILA
`int main() { D<int> d8; d8.m(); }`	COMPILA
`int main() { D<char> d9; d9.n(); }`	NON COMPILA
`int main() { D<Z> d10; d10.n(); }`	NON COMPILA
`int main() { D<char> d11; d11.o(); }`	COMPILA
`int main() { D<Z> d12; d12.o(); }`	NON COMPILA
`int main() { D<char> d13; d13.p(); }`	NON COMPILA
`int main() { D<Z> d14; d14.p(); }`	COMPILA