```
class A {
public:
  virtual ~A() {}
};
class B: public A {};
class C: virtual public B {};
class D: virtual public B {};
class E: public C, public D {};

char F(A* p, C& r) {
  B* punt = dynamic_cast<B*> (p);
  try{
    E& s = dynamic_cast<E&> (r);
  }
  catch(bad_cast) {
    if(punt) return 'O';
    else return 'M';
  }
  if(punt) return 'R';
  return 'A';
}

int main() {
  A a; B b; C c; D d; E e;
  cout << F(?,?) << F(?,?) << F(?,?) << F(?,?);
}
```

Definire opportunamente le chiamate nel main() usando gli oggetti locali a, b, c, d, e in modo tale che la sua esecuzione non provochi errori a run-time e produca in output la stampa ROMA.

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

Si consideri la gerarchia di classi per l'I/O. La classe base `ios` ha il distruttore virtuale, il costruttore di copia privato ed un unico costruttore (a 2 parametri con valori di default) protetto. Diciamo che le classi derivate da `istream` ma non da `ostream` (ad esempio `ifstream`), e `istream` stessa, sono *classi di input*, le classi derivate da `ostream` ma non da `istream` (ad esempio `ofstream`), ed `ostream` stessa, sono *classi di output*, mentre le classi derivate sia da `istream` che da `ostream` sono *classi di I/O* (esempi: `iostream` e `fstream`). Quindi ogni classe di input, output o I/O è una sottoclasse di `ios`. Definire una funzione `int F(ios& ref)` che restituisce -1 se il tipo dinamico di `ref` è un riferimento ad una classe di input, 1 se il tipo dinamico di `ref` è un riferimento ad una classe di output, 0 se il tipo dinamico di `ref` è un riferimento ad una classe di I/O, mentre in tutti gli altri casi ritorna 9.

Quindi, ad esempio, il seguente `main()` provoca la stampa riportata.

```
class D : public ios {
};

main() {
  istream& b = cin;
  ostream& c = cout;
  stringstream d;
  ifstream e("pippo");
  ofstream f("pluto");
  D g;
  cout << F(b) << ' ' << F(c) << ' ' << F(d) << ' ' << F(e) << ' '
       << F(f) << ' ' << F(g) <<  endl;
  // stampa: -1 1 0 -1 1 9
}
```

Definire un template di funzione `Fun(T1*, T2&)` che ritorna un booleano con il seguente comportamento. Consideriamo una istanziazione implicita `Fun(p,r)` dove supponiamo che i parametri di tipo `T1` e `T2` siano istanziati a tipi polimorfi (cioè che contengono almeno un metodo virtuale). Allora `Fun(p,r)` ritorna `true` se e soltanto se valgono le seguenti condizioni:

1. i parametri di tipo `T1` e `T2` sono istanziati allo stesso tipo;
2. siano `D1*` il tipo dinamico di `p` e `D2&` il tipo dinamico di `r`. Allora (i) `D1` e `D2` sono lo stesso tipo e (ii) questo tipo è un sottotipo proprio della classe `ios` della gerarchia di classi di I/O (si ricordi che `ios` è la classe base astratta della gerarchia).

Ad esempio, il seguente `main()` deve compilare e provocare le stampe indicate:

```
#include<iostream>
#include<fstream>
#include<typeinfo>
using namespace std;

class C { public: virtual ~C() {} };

main() {
  ifstream f("pippo"); fstream g("pluto"), h("zagor"); iostream* p = &h;
  C c1,c2;
  cout << Fun(&cout,cin) << endl; // stampa: 0
  cout << Fun(&cout,cerr) << endl; // stampa: 1
  cout << Fun(p,h) << endl; // stampa: 0
  cout << Fun(&f,*p) << endl; // stampa: 0
  cout << Fun(&g,h) << endl; // stampa: 1
  cout << Fun(&c1,c2) << endl; // stampa: 0
}
```

Primo appello scritto 2013-2014

Quesito 1

Si considerino le seguenti definizioni.

```
class Z {
private:
  int x;
};

class B {
private:
  Z x;
};

class D: public B {
private:
  Z y;
public:
  // ridefinizione di operator=
  ...
};
```

Ridefinire l'assegnazione `operator=` della classe `D` in modo tale che il suo comportamento coincida con quello dell'assegnazione standard di `D`.