```
class A {
public:
  virtual void m() =0;
};

class B: virtual public A {};

class C: virtual public A {
public:
  virtual void m() {}
};

class D: public B, public C {
public:
  virtual void m() {}
};

class E: public D {};

class F: public E {};

char G(A* p, B& r) {
  C* pc = dynamic_cast<E*>(&r);
  if(pc && typeid(*p)==typeid(r)) return 'G';
  if(!dynamic_cast<E*>(&r) && dynamic_cast<D*>(p)) return 'Z';
  if(!dynamic_cast<F*>(pc)) return 'A';
  else if(typeid(*p)==typeid(E)) return 'S';
  return 'E';
}
```

Si consideri inoltre il seguente statement.

```
  cout << G(new X1,*new Y1) << G(new X2,*new Y2) << G(new X3,*new Y3) << G(new X4,*new Y4)
       << G(new X5,*new Y5) << G(new X6,*new Y6) << G(new X7,*new Y7) << G(new X8,*new Y8);
```

Definire opportunamente le incognite di tipo `Xi` e `Yi` tra i tipi `A`, `B`, `C`, `D`, `E`, `F` della precedente gerarchia in modo tale che:

1. Lo statement non includa piú di una chiamata della funzione `G` con gli stessi parametri attuali
2. La compilazione dello statement non produca illegalità
3. L'esecuzione dello statement non provochi errori a run-time
4. L'esecuzione dello statement produca in output esattamente la stampa **SAGGEZZA**.