

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"); iosstream* 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
}
```

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).

```
template <class T1, class T2>
bool Fun(T1* p, T2& r) {
    return
        typeid(T1)==typeid(T2)  &&
        typeid(*p)==typeid(r)  &&
        dynamic_cast<ios*>(p) ;
}
```

Siano A, B, C e D distinte classi polimorfe. Si considerino le seguenti definizioni.

```
template<class X>
X& fun(X& ref) { return ref; };

main() {
    B b;
    fun<A>(b);
    B* p = new D();
    C c;
    try{
        dynamic_cast<B*>(fun<A>(c));
        cout << "topolino";
    }
    catch(bad_cast) { cout << "pippo "; }
    if( !(dynamic_cast<D*>(new B())) ) cout << "pluto ";
}
```

Si supponga che:

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

In tali ipotesi, per ognuna delle relazioni di sottotipo $X \leq Y$ nelle seguenti tabelle segnare con una croce l'entrata

- (a) "Vero" per indicare che X **sicuramente** è sottotipo di Y ;
 (b) "Falso" per indicare che X **sicuramente non** è sottotipo di Y ;
 (c) "Possibile" **altrimenti**, ovvero se non valgono nè (a) nè (b).

	Vero	Falso	Possibile
$A \leq B$			
$A \leq C$			
$A \leq D$			
$B \leq A$			
$B \leq C$			
$B \leq D$			

	Vero	Falso	Possibile
$C \leq A$			
$C \leq B$			
$C \leq D$			
$D \leq A$			
$D \leq B$			
$D \leq C$			

Siano A, B, C e D distinte classi polimorfe. Si considerino le seguenti definizioni.

```
template<class X>
X& fun(X& ref) { return ref; };

main() {
    B b;
    fun<A>(b);
    B* p = new D();
    C c;
    try{
        dynamic_cast<B&>(fun<A>(c));
        cout << "topolino";
    }
    catch(bad_cast) { cout << "pippo "; }
    if( !(dynamic_cast<D*>(new B())) ) cout << "pluto ";
}
```

Si supponga che:

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

In tali ipotesi, per ognuna delle relazioni di sottotipo $X \leq Y$ nelle seguenti tabelle segnare con una croce l'entrata

- (a) "Vero" per indicare che X **sicuramente** è sottotipo di Y ;
- (b) "Falso" per indicare che X **sicuramente non** è sottotipo di Y ;
- (c) "Possibile" **altrimenti**, ovvero se non valgono nè (a) nè (b).

Soluzione

vincoli:

$B \leq A$, $C \leq A$, $D \leq B$, $C \not\leq B$

	Vero	Falso	Possibile
$A \leq B$		X	
$A \leq C$		X	
$A \leq D$		X	
$B \leq A$	X		
$B \leq C$			X
$B \leq D$		X	

	Vero	Falso	Possibile
$C \leq A$	X		
$C \leq B$		X	
$C \leq D$		X	
$D \leq A$	X		
$D \leq B$	X		
$D \leq C$			X

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
}
```

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.

```
int F(ios& ref) {  
    if(dynamic_cast<istream*>(&ref) &&  
        !dynamic_cast<ostream*>(&ref)) return -1;  
    if(dynamic_cast<ostream*>(&ref) &&  
        !dynamic_cast<istream*>(&ref)) return 1;  
    if(dynamic_cast<istream*>(&ref) &&  
        dynamic_cast<ostream*>(&ref)) return 0;  
    return 9;  
}
```