Scrivere un programma consistente di esattamente tre classi `A`, `B` e `C` e della sola funzione `main()` che soddisfi le seguenti condizioni:

1. la classe `A` è definita come:

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

2. le classi `B` e `C` devono essere definite per ereditarietà e non contengono alcun membro
3. la funzione `main()` definisce le tre variabili:

   ```
   A* pa = new A; B* pb = new B; C* pc = new C;
   ```

   e nessuna altra variabile (di alcun tipo)
4. la funzione `main()` può **utilizzare solamente** espressioni di tipo `A*`, `B*` e `C*`, **non può** sollevare eccezioni mediante una `throw` e **non può** invocare l'operatore `new`
5. il programma deve compilare correttamente
6. l'esecuzione di `main()` **deve provocare un errore run-time**.

```
class A{
        public:
        virtual ~A(){};
};
class B: public A{};
class C: public A{};
int main(){
        A* pa = new A;
        B* pb = new B;
        C* pc = new C;
}
```

```
// ERRORE RUNTIME
auto c1 = dynamic_cast<C&>(pb);
OPPURE
auto b1 = dynamic_cast<b&>(pc);
```


STATIC_CAST (B&) (PB->NC()) ->MC();

A & NON C().

ANALOGO = WORKS ONLY STATIC_CAST

```
class B: public A {};
class C: public A {};
int main() {/* ...*/ dynamic_cast<C&>(*pb);}
```

## Soluzione

Dereferencing a NULL pointer is undefined behavior.

In fact the standard calls this exact situation out in a note (8.3.2/4 "References"):

> Note: in particular, a null reference cannot exist in a well-defined program, because the only way to create such a reference would be to bind it to the "object" obtained by dereferencing a null pointer, which causes undefined behavior.

```
class Z {
public:
  ...
};

template <class T1, class T2=Z>
class C {
public:
  T1 x;
  T2* p;
};

template<class T1,class T2>
void fun(C<T1,T2>* q) {
  ++(q->p);
  if(true == false) cout << ++(q->x);
  else cout << q->p;
  (q->x)++;
  if(*(q->p) == q->x) *(q->p) = q->x;
  T1* ptr = &(q->x);
  T2 t2 = q->x;
}

main(){
  C<Z> c1; fun(&c1); C<int> c2; fun(&c2);
}
```

++ POSTFISSO = INT operator++(int);

OSTREAM& operator<<(const OSTREAM&, T2);

T2 può essere Z => serve ++

Z(const Z&);

int operator(int). -> POSTFISSO

FITTIZIO PER DISTINGUERLO DA PREFIX

bool operator==(const Z&);

T2(Z);

Si considerino le precedenti definizioni. Fornire una dichiarazione (non è richiesta la definizione) dei membri pubblici della classe `Z` nel **minor numero possibile** in modo tale che la compilazione del precedente `main()` non produca errori. **Attenzione:** ogni dichiarazione in `Z` non necessaria per la corretta compilazione del `main()` sarà penalizzata.