La ragione per cui `typeid(pb3) == typeid(F)` non stampa come ti aspetti è legata al fatto che stai controllando il tipo del **puntatore** `pb3` e non del tipo dell'**oggetto** a cui punta.

Nel tuo codice hai:

```
B* pb3 = new F();
```

Il problema è nella riga 06 del tuo esercizio:

```
if(typeid(pb3) == typeid(F)) pb3->f(Z(2));
```

Spiegazione del comportamento:

1. `typeid(pb3)` restituisce il tipo **statico** del puntatore, che è `B*` (puntatore a B)
2. `typeid(F)` restituisce il tipo della classe `F`
3. Questi due tipi sono diversi, quindi il confronto fallisce

Soluzione corretta:

Per ottenere il tipo dinamico dell'oggetto puntato, devi **dereferenziare** il puntatore:

```
if(typeid(*pb3) == typeid(F)) pb3->f(Z(2));
```

Regola generale per `typeid`:

- `typeid(puntatore)` → restituisce sempre il tipo statico del puntatore
- `typeid(*puntatore)` → se la classe è polimorfa (ha almeno un metodo virtuale), restituisce il tipo dinamico dell'oggetto puntato
- `typeid(riferimento)` → se la classe è polimorfa, restituisce il tipo dinamico dell'oggetto riferito

Nel tuo caso, dato che la classe B ha metodi virtuali (è una classe polimorfa), `typeid(*pb3)` restituirà correttamente `F`, permettendo al confronto di avere successo e alla stampa di essere eseguita.

Sì, **corretto**! Se avessimo:

```
template <class T>
T* Fun(T& ref) { ... }
```

allora `Fun(Fun(pb1))` stamperebbe **"B B"**.

Spiegazione:

Situazione attuale (con `T* Fun(T* pt)`):

1. `Fun(pb1)`: `pb1` è `B*` → chiama `Fun<B>(B* pt)` → stampa "B" → **ritorna** `B*`
2. `Fun(risultato)`: Il risultato è `B*` (puntatore) → può chiamare di nuovo `Fun<B>(B* pt)` → ma internamente può accedere all'oggetto base `A` → stampa "A"

Con `T* Fun(T& ref)`:

1. `Fun(pb1)`: `pb1` è `B*`, ma la funzione si aspetta `T&` (riferimento)
   - Per chiamare `Fun(T& ref)` con un puntatore, il puntatore deve essere **dereferenziato**
   - Quindi si chiama `Fun(*pb1)` → `*pb1` è di tipo `B&` → stampa "B" → **ritorna** `B*`
2. `Fun(risultato)`: Il risultato è `B*` (puntatore)
   - Di nuovo, per passare a `Fun(T& ref)`, deve essere dereferenziato: `Fun(*risultato)`
   - `*risultato` è di tipo `B&` → stampa "B"

Risultato: "B B"

La chiave è che con `T& ref` (riferimento), il compilatore deve **dereferenziare** i puntatori per ottenere il riferimento, mantenendo così il tipo `B` in entrambe le chiamate, invece di permettere il "slicing" verso il tipo base `A`.

Questo comportamento deriva dalle **regole di deduzione dei template** dove:

- `T*` accetta puntatori direttamente
- `T&` richiede riferimenti, forzando la dereferenziazione dei puntatori