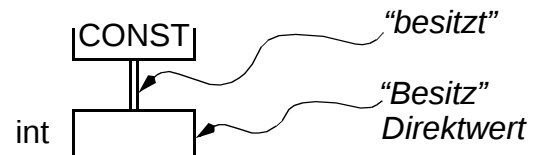


ARBEIT MIT POINTERN (ZEIGERVARIABLEN)

(1) DREI ARTEN DER DATENHALTUNG

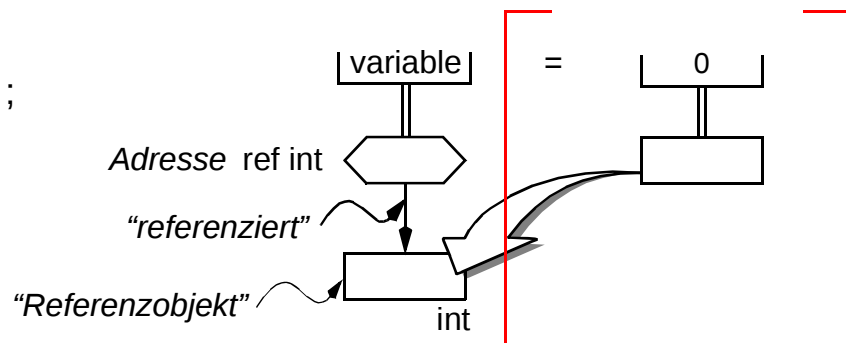
□ Konstante

```
final int CONST = 10 ;
```



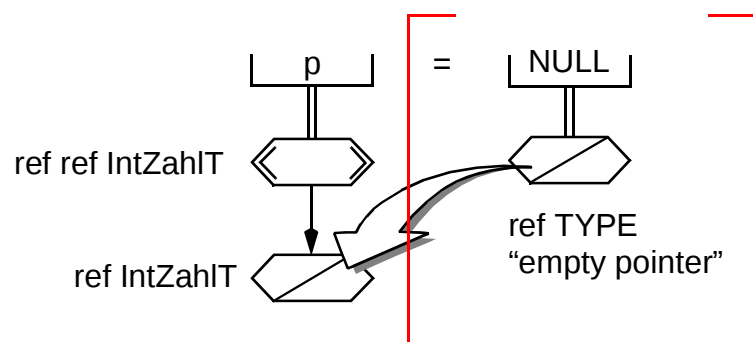
□ Variable

```
int variable [ = 0 ] ;
```



□ Pointer (Zeigervariable, Adreßvariable)

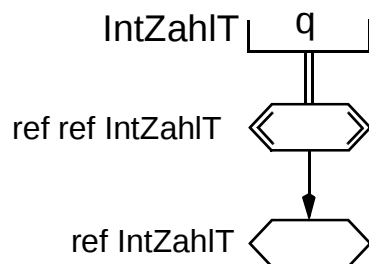
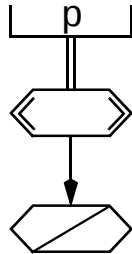
```
class IntZahlT { int i } ;  
IntZahlT p [ = NULL ] ;
```



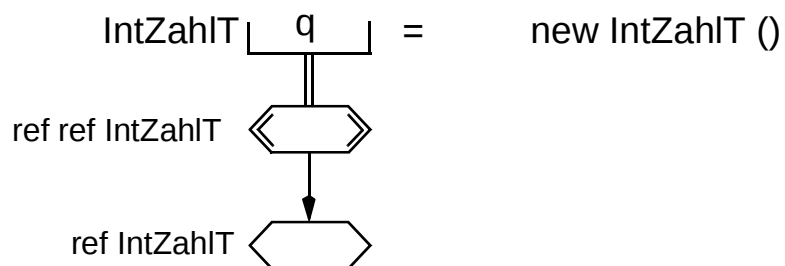
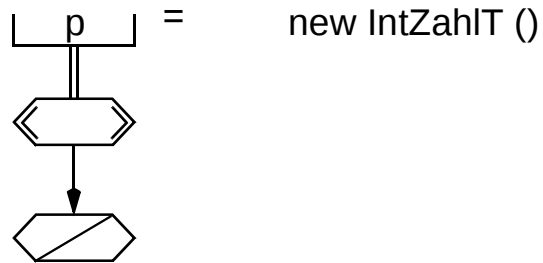
Anm.:

- ein auf NULL gesetzter Pointer kann offensichtlich nicht (sinnvoll) entreferenziert werden;
- der Versuch des Entreferenzierens führt zu einem Laufzeitfehler;

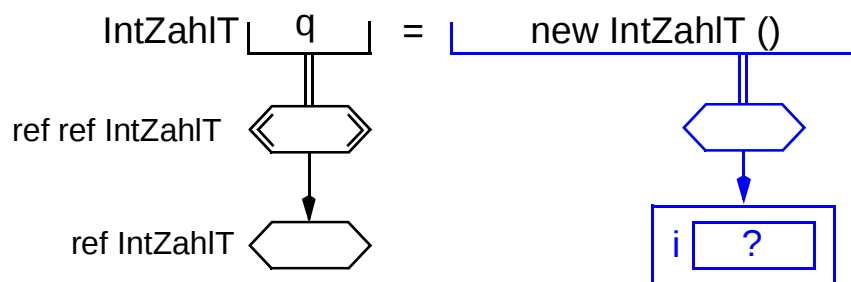
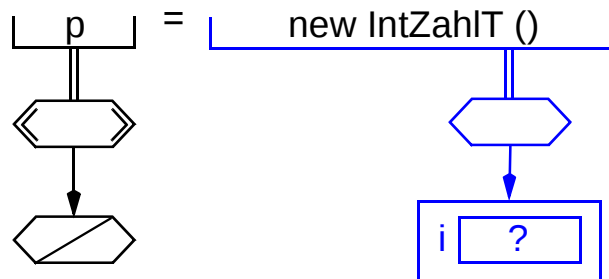
(2) ZUWEISEN VON (GLOBALEM) SPEICHER



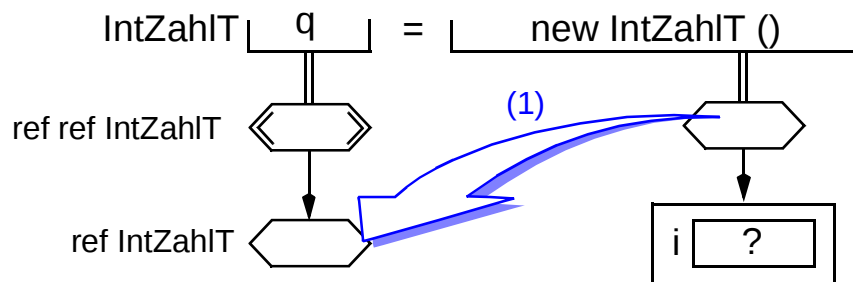
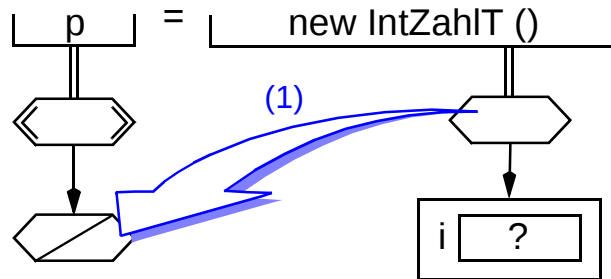
(2) ZUWEISEN VON (GLOBALEM) SPEICHER



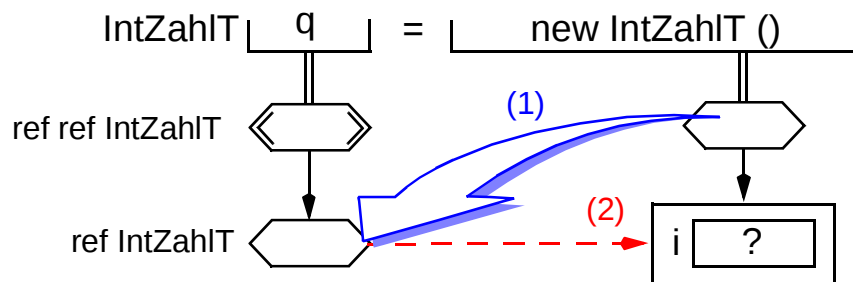
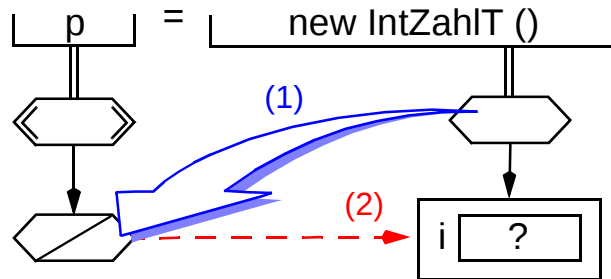
(2) ZUWEISEN VON (GLOBALEM) SPEICHER



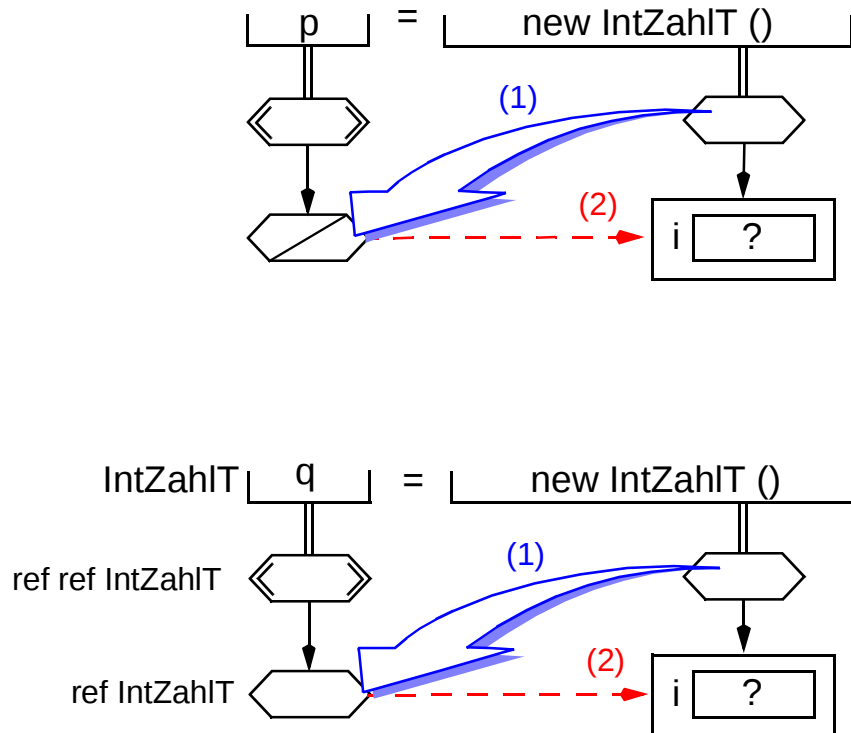
(2) ZUWEISEN VON (GLOBALEM) SPEICHER



(2) ZUWEISEN VON (GLOBALEM) SPEICHER



(2) ZUWEISEN VON (GLOBALEM) SPEICHER



Anm.:

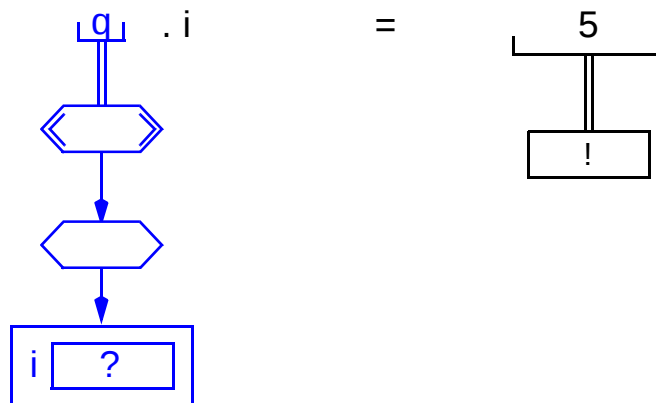
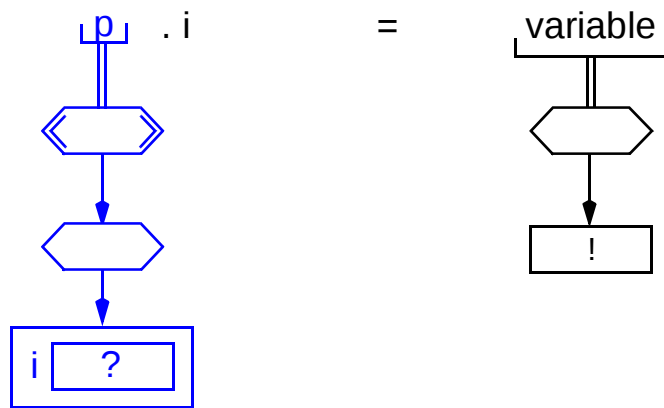
- denkbare Systemreaktionen, falls kein Speicher mehr frei:
 - NULL-Zuweisung (Assert-Anweisung (`p != NULL`))
 - Fehlermeldung durch das Laufzeitsystem
 - BS-Fehlermeldung (segmentation fault, core dumped, ...)
- Ziel einer Wertzuweisung ist immer das Referenzobjekt der linken Seite.**
 - > vgl. hierzu auch (3) und (4)

(3) ENTREFERENZIEREN

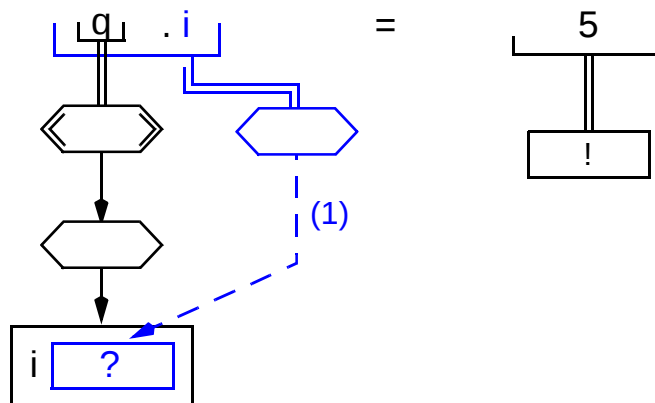
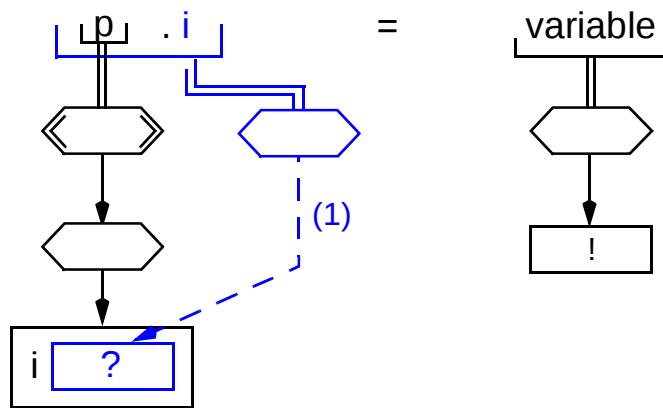
`p[i] = variable`

`q[i] = 5`

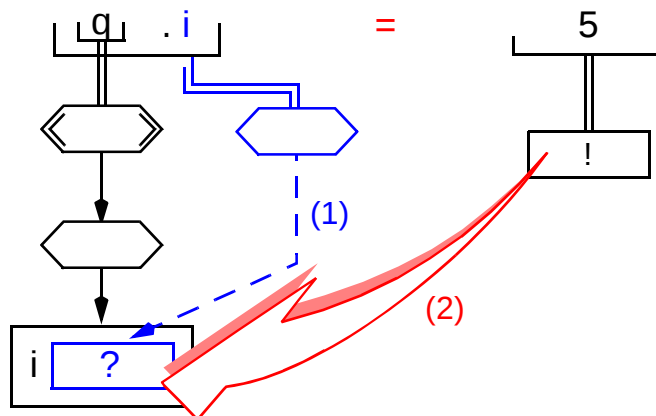
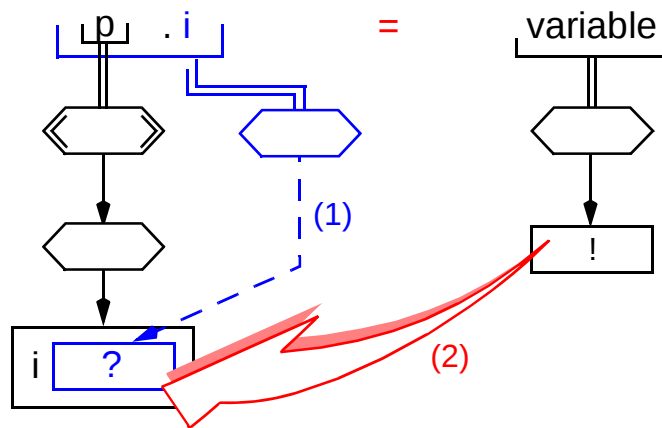
(3) ENTREFERENZIEREN



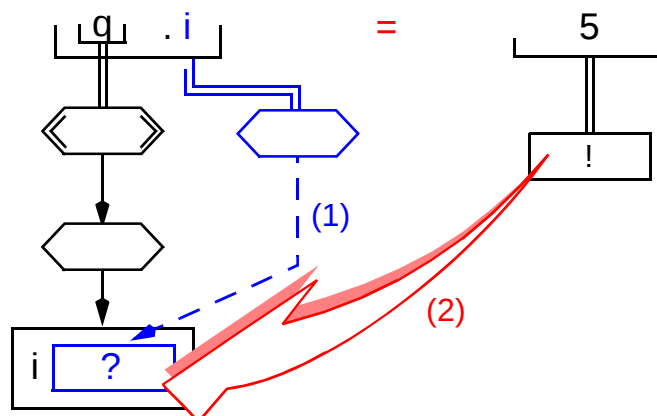
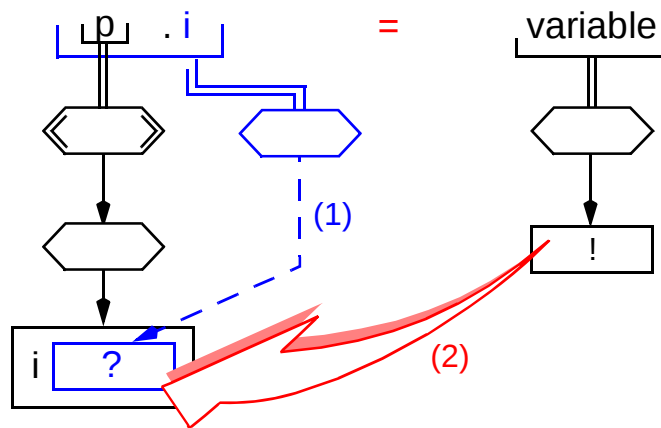
(3) ENTREFERENZIEREN



(3) ENTREFERENZIEREN



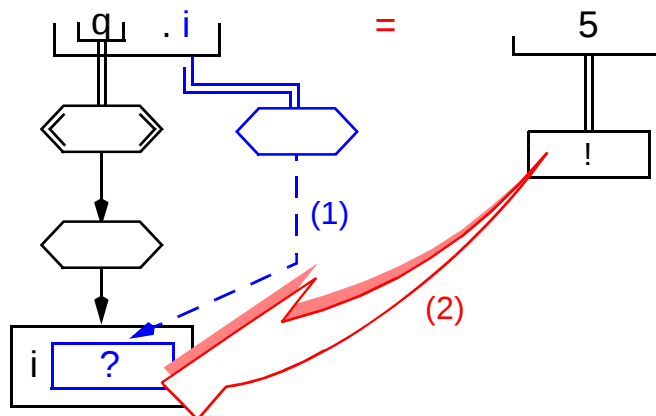
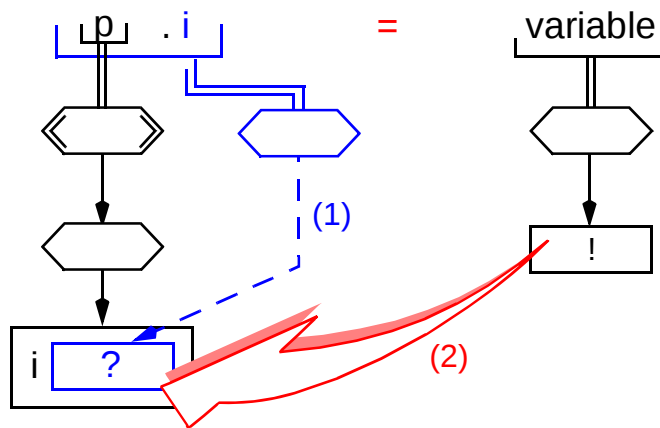
(3) ENTREFERENZIEREN



Anm.:

- `print ( p.i, q.i)`

(3) ENTREFERENZIEREN



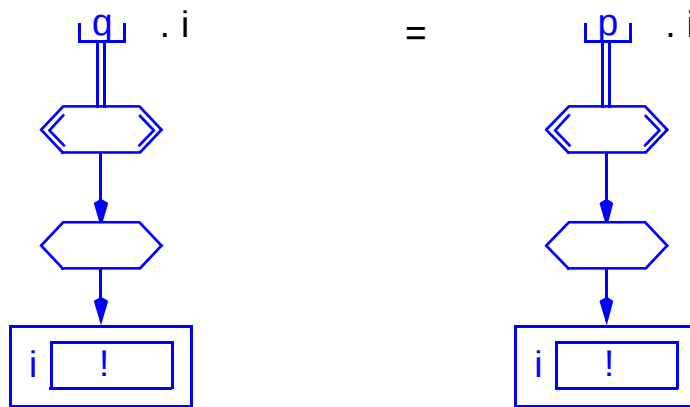
Anm.:

- `print ( p.i, q.i) -> 0 5`

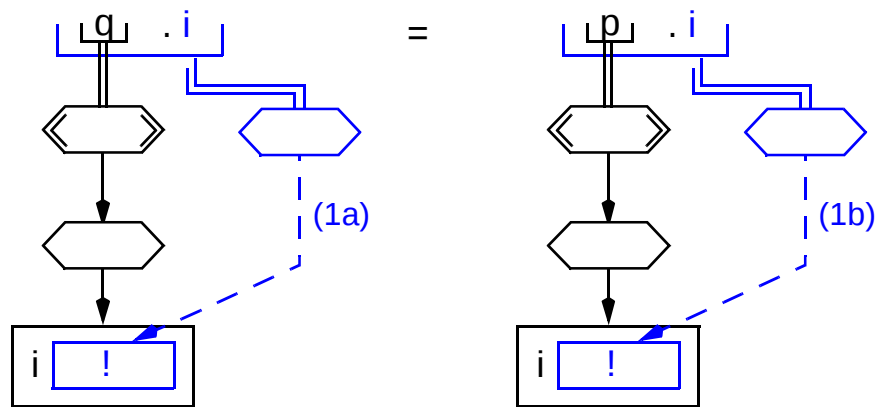
(4) ZUWEISUNGEN MIT POINTERN

`q[i] = p[i]`

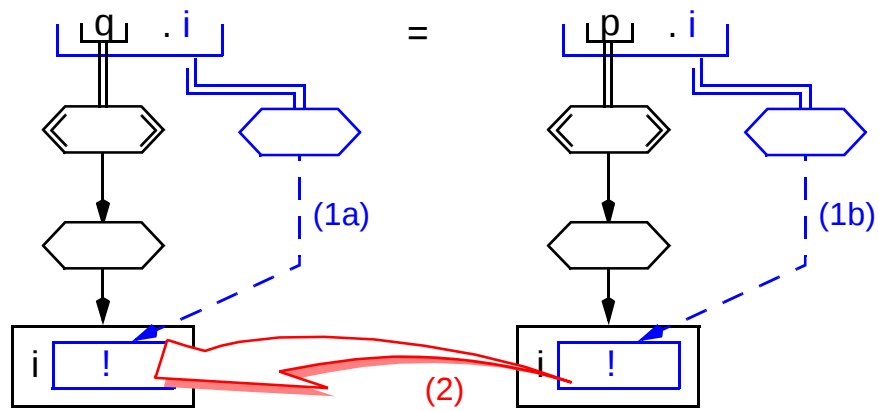
(4) ZUWEISUNGEN MIT POINTERN



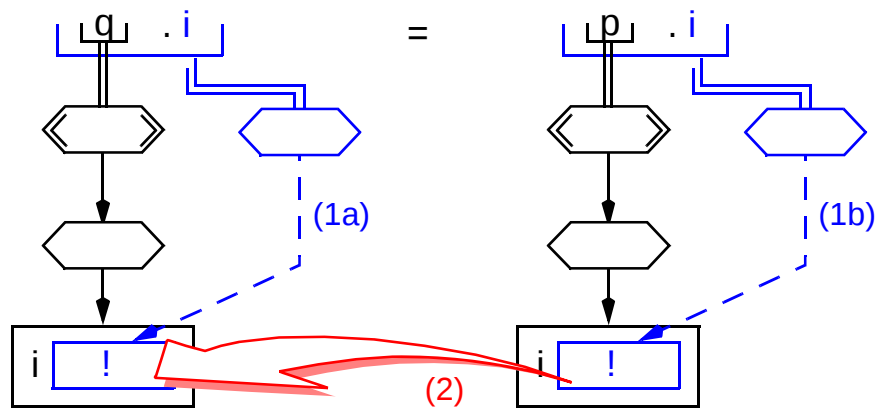
(4) ZUWEISUNGEN MIT POINTERN



(4) ZUWEISUNGEN MIT POINTERN



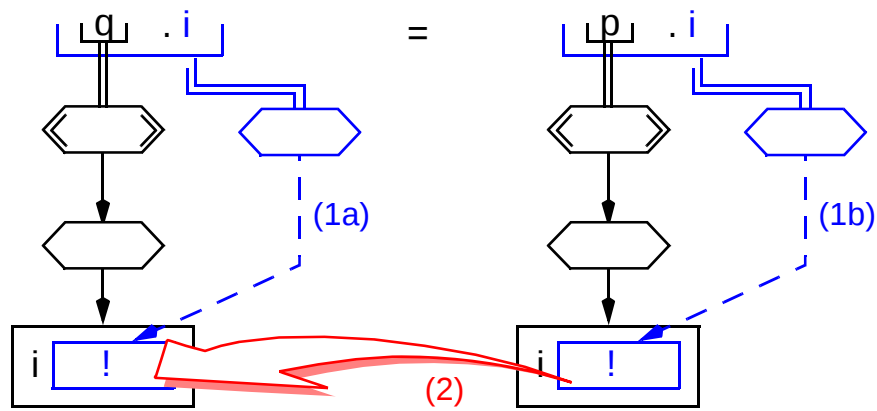
(4) ZUWEISUNGEN MIT POINTERN



Anm.:

- `print (p.i, q.i)`

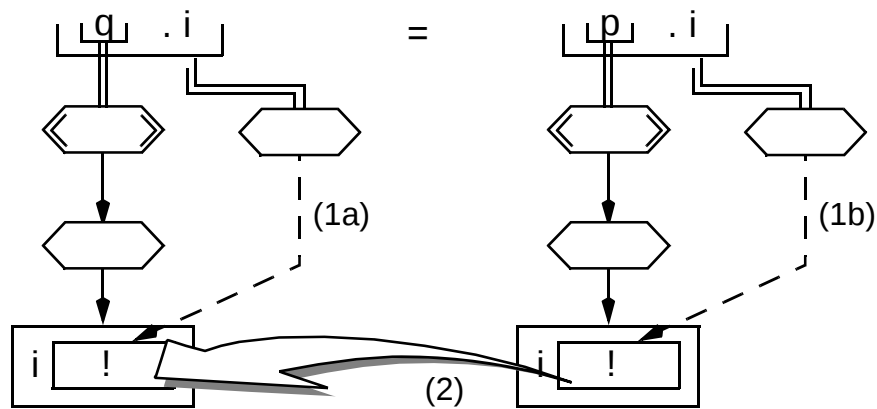
(4) ZUWEISUNGEN MIT POINTERN



Anm.:

- `print (p.i, q.i) -> 0 0`

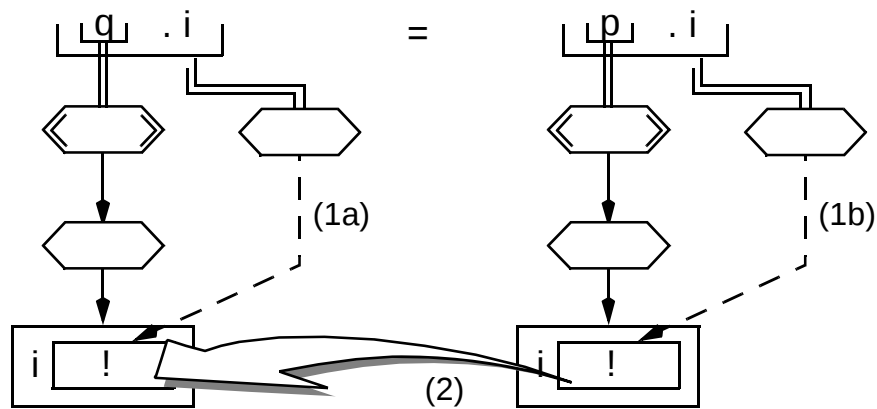
(4) ZUWEISUNGEN MIT POINTERN



Anm.:

- `print (p.i, q.i) -> 0 0`
- Wenn nun `p` neue Direktwerte zugewiesen werden, sind über `q` weiter die alten Direktwerte verfügbar.
z. Bsp. `p.i = 7; print (p.i, q.i) -> 7 0`

(4) ZUWEISUNGEN MIT POINTERN

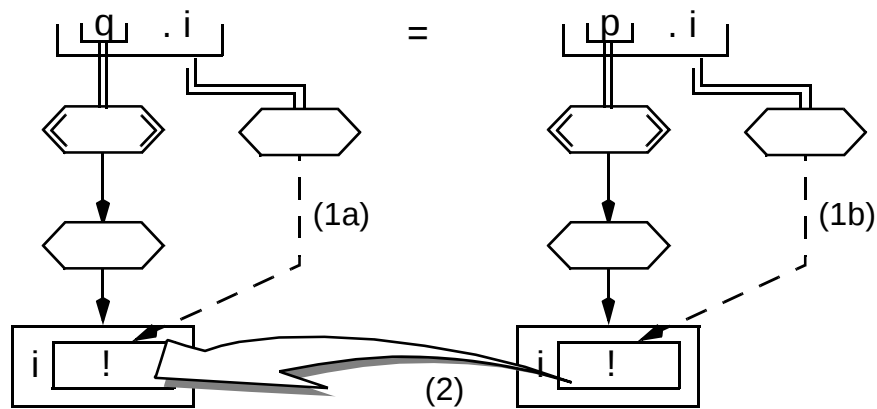


Anm.:

- `print (p.i, q.i) -> 0 0`
- Wenn nun p neue Direktwerte zugewiesen werden, sind über q weiter die alten Direktwerte verfügbar.
z. Bsp. `p.i = 7; print (p.i, q.i) -> 7 0`

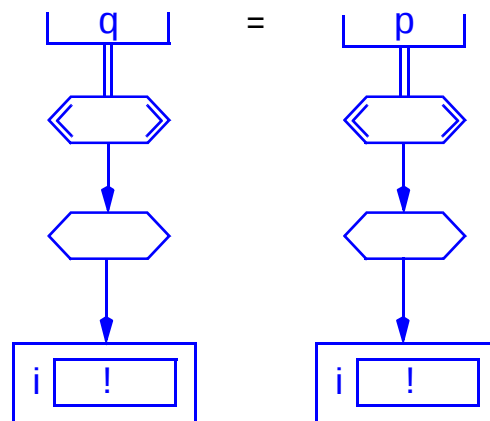
q = p

(4) ZUWEISUNGEN MIT POINTERN

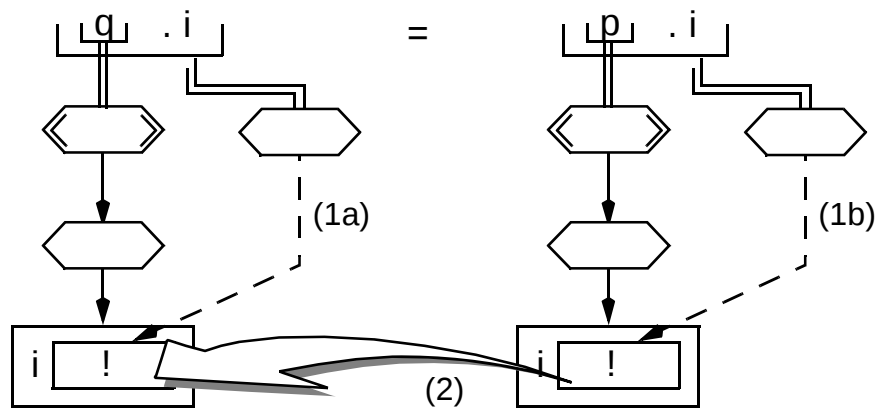


Anm.:

- `print (p.i, q.i) -> 0 0`
- Wenn nun p neue Direktwerte zugewiesen werden, sind über q weiter die alten Direktwerte verfügbar.
z. Bsp. `p.i = 7; print (p.i, q.i) -> 7 0`

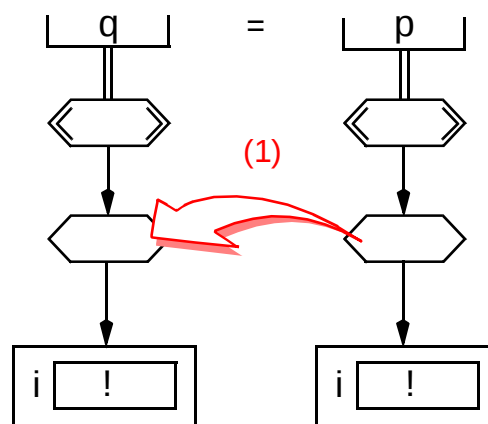


(4) ZUWEISUNGEN MIT POINTERN

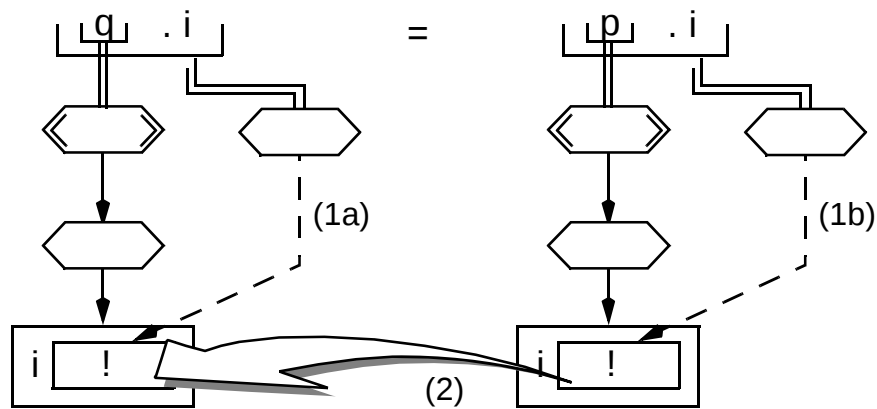


Anm.:

- `print (p.i, q.i) -> 0 0`
- Wenn nun `p` neue Direktwerte zugewiesen werden, sind über `q` weiter die alten Direktwerte verfügbar.
z. Bsp. `p.i = 7; print (p.i, q.i) -> 7 0`

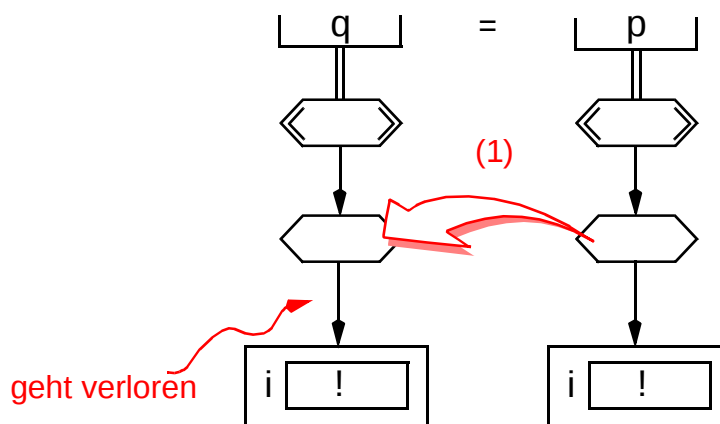


(4) ZUWEISUNGEN MIT POINTERN

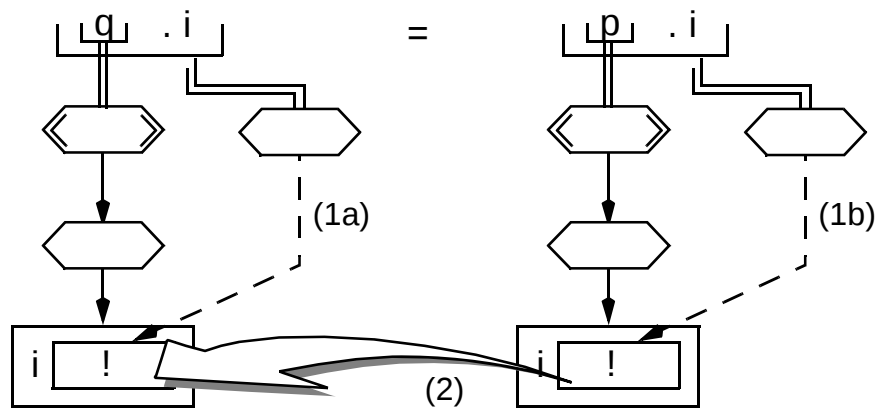


Anm.:

- `print (p.i, q.i) -> 0 0`
- Wenn nun p neue Direktwerte zugewiesen werden, sind über q weiter die alten Direktwerte verfügbar.
z. Bsp. `p.i = 7; print (p.i, q.i) -> 7 0`

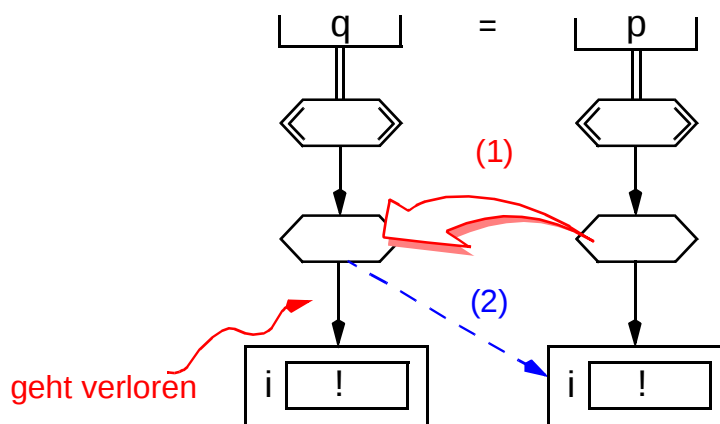


(4) ZUWEISUNGEN MIT POINTERN

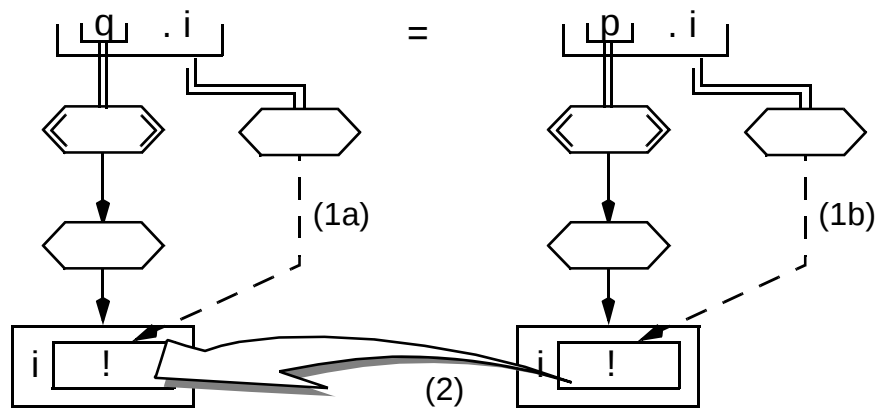


Anm.:

- `print (p.i, q.i) -> 0 0`
- Wenn nun p neue Direktwerte zugewiesen werden, sind über q weiter die alten Direktwerte verfügbar.
z. Bsp. `p.i = 7; print (p.i, q.i) -> 7 0`

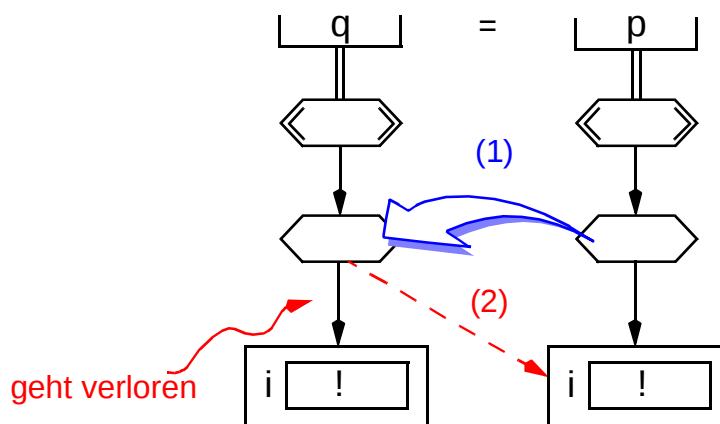


(4) ZUWEISUNGEN MIT POINTERN



Anm.:

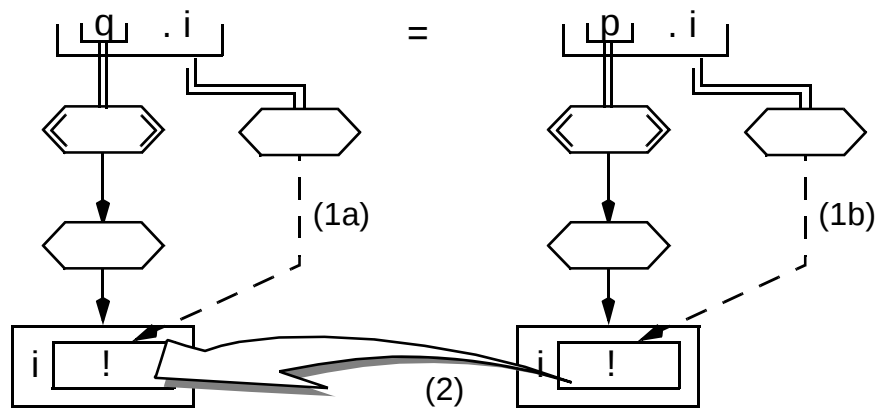
- `print (p.i, q.i) -> 0 0`
- Wenn nun p neue Direktwerte zugewiesen werden, sind über q weiter die alten Direktwerte verfügbar.
z. Bsp. `p.i = 7; print (p.i, q.i) -> 7 0`



Anm.:

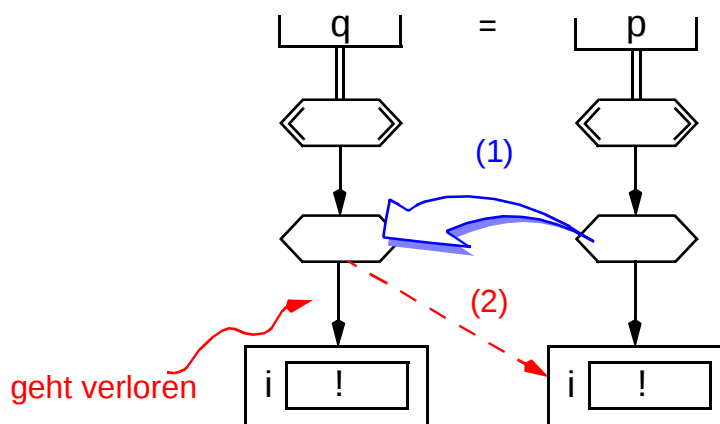
- `print (p.i, q.i)`

(4) ZUWEISUNGEN MIT POINTERN



Anm.:

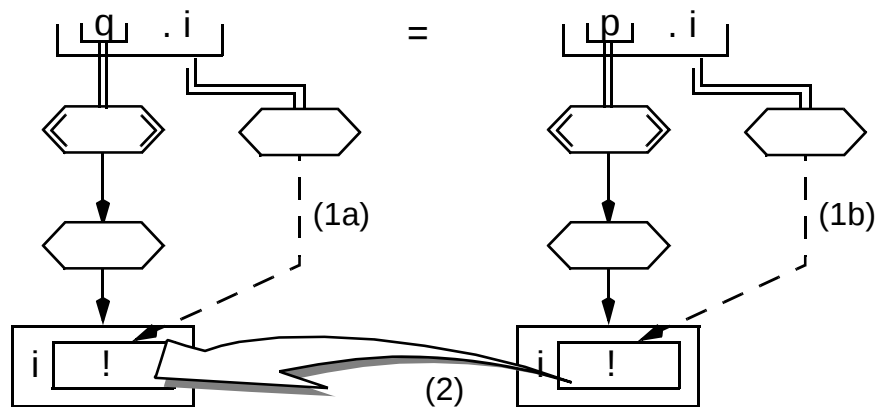
- `print (p.i, q.i) -> 0 0`
- Wenn nun p neue Direktwerte zugewiesen werden, sind über q weiter die alten Direktwerte verfügbar.
z. Bsp. `p.i = 7; print (p.i, q.i) -> 7 0`



Anm.:

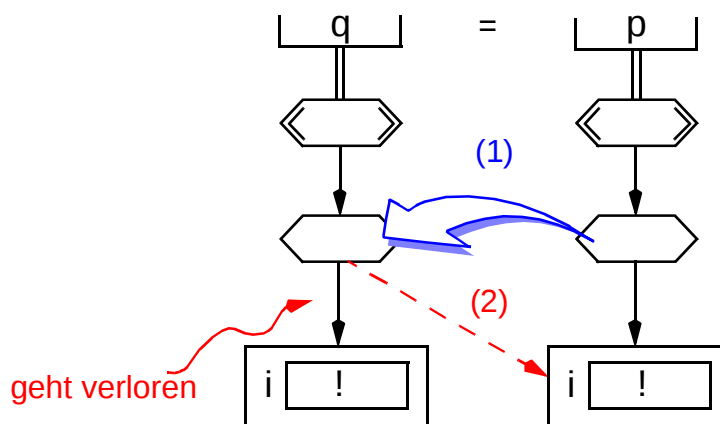
- `print (p.i, q.i) -> 7 7`

(4) ZUWEISUNGEN MIT POINTERN



Anm.:

- `print (p.i, q.i) -> 0 0`
- Wenn nun `p` neue Direktwerte zugewiesen werden, sind über `q` weiter die alten Direktwerte verfügbar.
z. Bsp. `p.i = 7; print (p.i, q.i) -> 7 0`



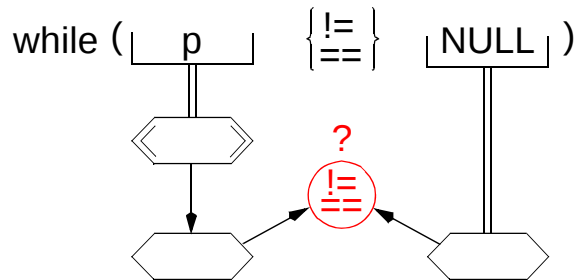
Anm.:

- `print (p.i, q.i) -> 7 7`
- Wenn nun `p` neue Direktwerte zugewiesen werden, sind diese auch über `q` erreichbar, et vice versa.
z. Bsp. `p.i = 3; print (p.i, q.i) -> 3 3`

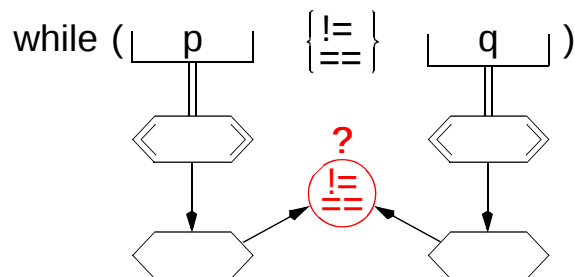
(5) OPERATIONEN AUF POINTERN (ADRESSEN)

(5A) TEST AUF NULL

(MIT ALLEN ADREßTYPEN VERTRÄGLICH)



(5B) TEST AUF GLEICHHEIT/UNGLEICHHEIT, FALLS POINTER VOM GLEICHEN TYP



(5C) WEITERE OPERATIONEN WERDEN NICHT BENÖTIGT!

(6) SPEICHERFREIGABE

→ Voraussetzung für Wiederverwendbarkeit des globalen Speichers

p = NULL; /* simuliert Freigabe, analog zu einem free(p) */

q = NULL;

(6) EIN DEMO-PROGRAMM

```
class PointerDemo {

    // Umgang mit Pointern, d.h. Zeigervariablen
    public static void main(String[] args) {
        /*a1*/ class MyInteger{int i;};
            MyInteger p = null;
            p = new MyInteger();

        /*a2*/ MyInteger q = new MyInteger();
            int variable = 5;

        /*b*/ p.i = 3;
            q.i = variable;
            System.out.println("(b): p=> "+p.i+"  q=> "+q.i);
            System.out.println(" (p==q) => "+(p==q)+"\n");

        /*c*/ p.i = q.i;
            System.out.println("(c): p=> "+p.i+"  q=> "+q.i);
            System.out.println(" (p==q) => "+(p==q)+"\n");

        /*d*/ p = q;
            System.out.println("(d): p=> "+p.i+"  q=> "+q.i);
            System.out.println(" (p==q) => "+(p==q)+"\n");

        /*e*/ p.i = 7;
            System.out.println("(e): p=> "+p.i+"  q=> "+q.i);
            System.out.println(" (p==q) => "+(p==q)+"\n");

    } // main

} //PointerDemo
```