Zeiger und Arrays

☑ **Lösung zu Aufgabe 2.1** Der repräsentierte Wert kann jeweils wie folgt verändert werden.

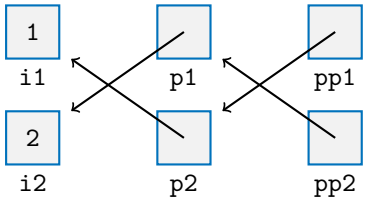
```
a = 4; *b = 4; *c = 4;
```

Bei `a` befindet sich der repräsentierte Wert direkt auf dem sogenannten *Stack*. Bei `b` und `c` befindet er sich hingegen auf dem sogenannten *Heap*.

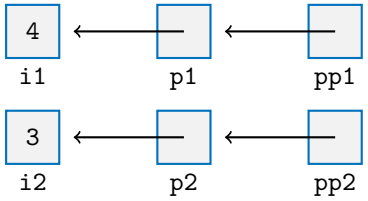
Sobald die Ausführung der Funktion `f` abgeschlossen ist, wird der von ihr verwendete Speicher auf dem Stack automatisch aufgeräumt und deshalb wird auch der von `a` verwendete Speicher automatisch freigegeben. Der vom Zeiger `b` referenzierte Speicher auf dem Heap wird jedoch nicht automatisch aufgeräumt, was zu einem sogenannten *Memory leak* führt (dies könnte man aber mit der expliziten Anweisung `delete b;` verhindern). Obwohl sich der von `c` referenzierte Speicher auch auf dem Heap befindet, wird durch eine clevere Implementierung der Klasse `std::unique_ptr` sichergestellt, dass dieser Speicher auch automatisch freigegeben wird, sobald die Funktion `f` fertig ist.

☑ **Lösung zu Aufgabe 2.2** Die erhaltene Ausgabe ist `1 2;2 1;1 2;4 3;4 3;4 3;`, wobei das Zeichen `;` einen Zeilenumbruch repräsentieren soll. Auch wenn Ihnen der Code von der Aufgabenstellung vielleicht zurecht ein bisschen künstlich vorkommt, lohnt es sich trotzdem, einmal ein solches Beispiel durchzuarbeiten. Dabei stellt es sich oft als hilfreich heraus, sich die Situation mit all den Zeigern (und Zeigern auf Zeiger) schematisch aufzuzeichnen.

Beispielsweise sind die Zeiger gerade nach Ausführen von Zeile 7 auf folgende Art und Weise verschränkt, was relativ anschaulich den ersten Teil `1 2;2 1;1 2;` der erhaltenen Ausgabe erklärt:



Auf Zeile 10 wird deshalb über den Zeiger `p1` indirekt die Variable `i2` auf den Wert `3` gesetzt, während die Variable `i1` über `p2` auf den Wert `4` gesetzt wird. Auf analoge Weise wird dann auf Zeile 13 über die Zeiger `pp1` und `pp2` indirekt dafür gesorgt, dass `p1` auf `i1` zeigt und `p2` auf `i2`. Auf Zeile 16 werden schliesslich auch noch die Zeiger `pp1` und `pp2` geradegebogen. Ganz am Schluss sieht die Situation deshalb wie folgt aus, was den zweiten Teil `4 3;4 3;4 3;` der erhaltenen Ausgabe erklärt:



☑ **Lösung zu Aufgabe 2.3** Wir verwenden in folgendem Code den Zeiger `greatest`, um uns die Adresse der bisher grössten gefundenen Zahl zu merken. Dann gehen wir mit einer Schleife über alle Positionen im übergebenen Array und aktualisieren `greatest` immer dann, wenn wir eine noch grössere Zahl finden.

```
int* max(int* first, size_t length) {
  int* greatest = first;
  for(int* i = first + 1; i < first + length; ++i) {
    if(*i > *greatest) {
      greatest = i;
    }
  }
  return greatest;
}
```