Beispiel (Forts.): Bestimmung des Minimums

```
int result = a[0];
int i = 1;          // Initialisierung
while (i < a.length) {
    if (a[i] < result)
        result = a[i];
    i = i+1;   // Modifizierung
    }
write(result);
```

Mithilfe des `for`-Statements:

```
int result = a[0];
for (int i = 1; i < a.length; ++i)
    if (a[i] < result)
        result = a[i];
write(result);
```

Allgemein:

`for` (init; cond; modify) stmt

... entspricht:

{ init ; `while` (cond) { stmt modify ;} }

... wobei `++i` äquivalent ist zu `i = i+1` .

Warnung:

- Die Zuweisung `x = x-1` ist in Wahrheit ein Ausdruck.
- Der Wert ist der Wert der rechten Seite.
- Die Modifizierung der Variable `x` erfolgt als Seiteneffekt.
- Der Semikolon “;” hinter einem Ausdruck wirft nur den Wert weg.

$\Longrightarrow$ ... fatal für Fehler in Bedingungen ...

```
boolean x = false;
if (x = true)
   write("Sorry! This must be an error ...");
```

- Die Operatoranwendungen `++x` und `x++` inkrementieren beide den Wert der Variablen `x`.
- `++x` tut das, bevor der Wert des Ausdrucks ermittelt wird (Pre-Increment).
- `x++` tut das, nachdem der Wert ermittelt wurde (Post-Increment).
- `a[x++] = 7;` entspricht:

```
a[x] = 7;
x = x+1;
```

- `a[++x] = 7;` entspricht:

```
x = x+1;
a[x] = 7;
```

5.6 Funktionen und Prozeduren

Oft möchte man

- Teilprobleme separat lösen; und dann
- die Lösung mehrfach verwenden.

Beispiel: Einlesen eines Felds

```
public static int[] readArray(int number) {
       // number = Anzahl der zu lesenden Elemente
    int[] result = new int[number]; // Anlegen des Felds
    for (int i = 0; i < number; ++i) {
        result[i] = read();
        }
    return result;
}
```

- Die erste Zeile ist der Header der Funktion.
- `public` sagt, wo die Funktion verwendet werden darf (↑kommt später)
- `static` kommt ebenfalls später.
- `int[]` gibt den Typ des Rückgabe-Werts an.
- `readArray` ist der Name, mit dem die Funktion aufgerufen wird.
- Dann folgt (in runden Klammern und komma-separiert) die Liste der formalen Parameter, hier: `(int number)`.
- Der Rumpf der Funktion steht in geschwungenen Klammern.
- `return` expr beendet die Ausführung der Funktion und liefert den Wert von expr zurück.

- Die Variablen, die innerhalb eines Blocks angelegt werden, d.h. innerhalb von "`{`" und "`}`", sind nur innerhalb dieses Blocks sichtbar, d.h. benutzbar (lokale Variablen).
- Der Rumpf einer Funktion ist ein Block.
- Die formalen Parameter können auch als lokale Variablen aufgefasst werden.
- Bei dem Aufruf `readArray(7)` erhält der formale Parameter `number` den Wert `7`.

Weiteres Beispiel: Bestimmung des Minimums

```
public static int min (int[] b) {
   int result = b[0];
   for (int i = 1; i < b.length; ++i) {
       if (b[i] < result)
           result = b[i];
       }
   return result;
   }
```

... daraus basteln wir das Java-Programm `Min` :

```
public class Min extends MiniJava {
   public static int[] readArray (int number) { ... }
   public static int min (int[] b) { ... }
      // Jetzt kommt das Hauptprogramm
   public static void main (String[] args) {
      int n = read();
      int[] a = readArray(n);
      int result =  min(a);
      write(result);
      }    // end of main()
}          // end of class Min
```

- Manche Funktionen, deren Ergebnistyp `void` ist, geben gar keine Werte zurück – im Beispiel: `write()` und `main()`. Diese Funktionen heißen Prozeduren.
- Das Hauptprogramm hat immer als Parameter ein Feld `args` von `String`-Elementen.
- In diesem Argument-Feld werden dem Programm Kommandozeilen-Argumente verfügbar gemacht.

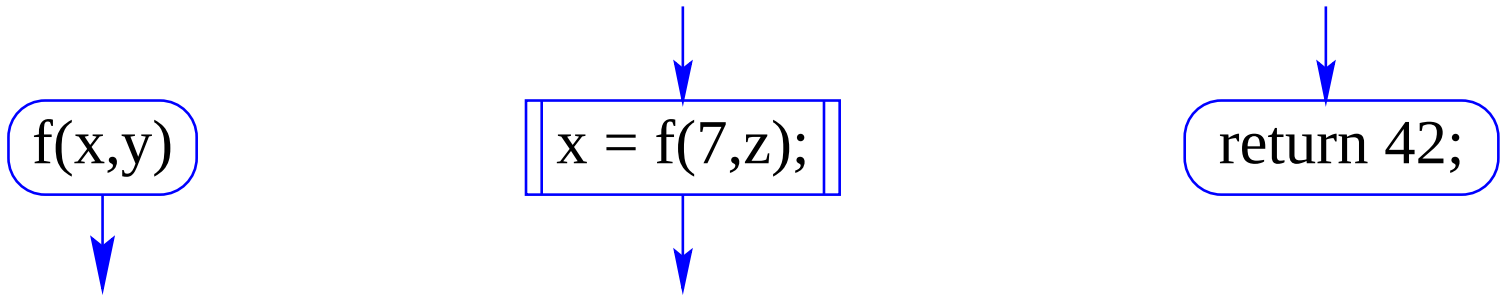
```
public class Test extends MiniJava {
   public static void main (String [] args) {
      write(args[0]+args[1]);
      }
}  // end of class Test
```

Dann liefert der Aufruf:

```
java Test "Hel" "lo World!"
```

... die Ausgabe: Hello World!

Um die Arbeitsweise von Funktionen zu veranschaulichen, erweitern/modifizieren wir die Kontrollfluss-Diagramme:



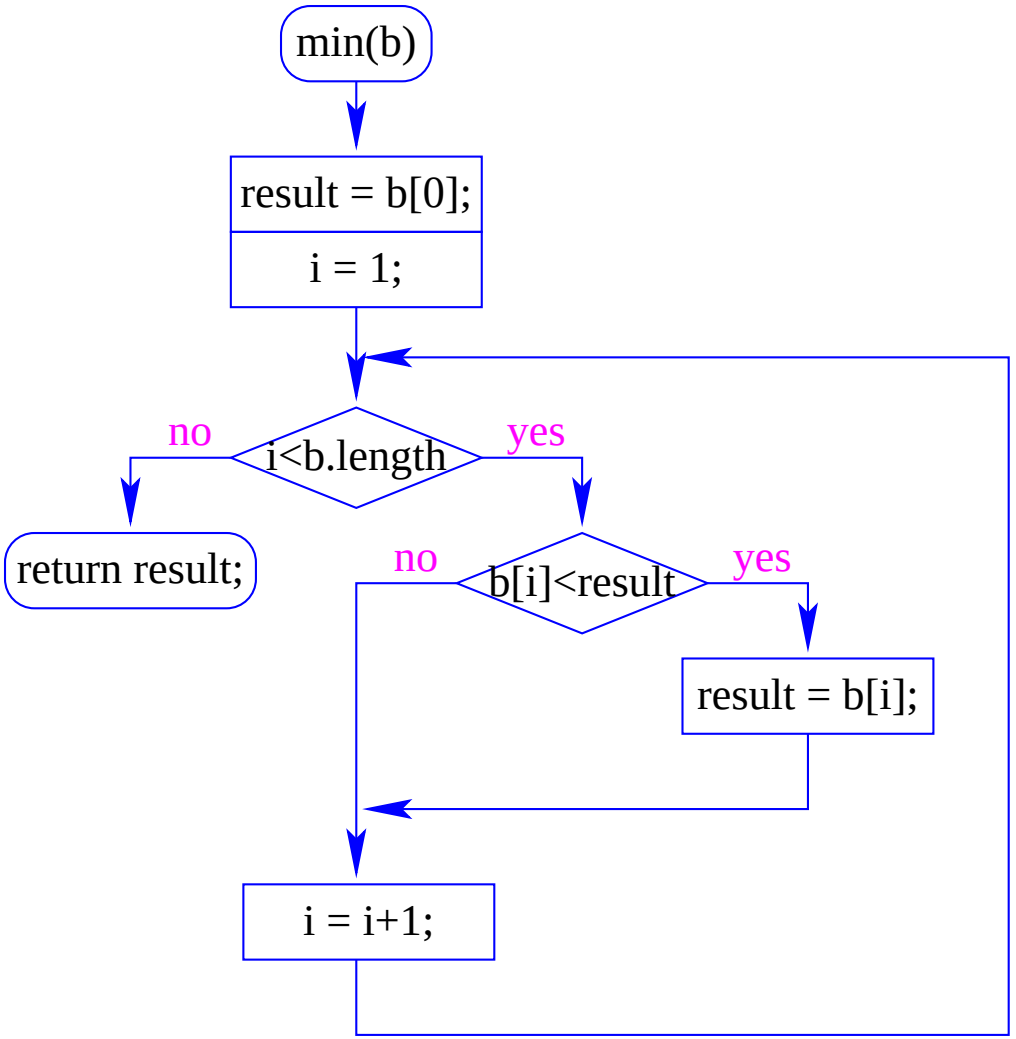
Startknoten der Funktion f

Funktionsaufruf

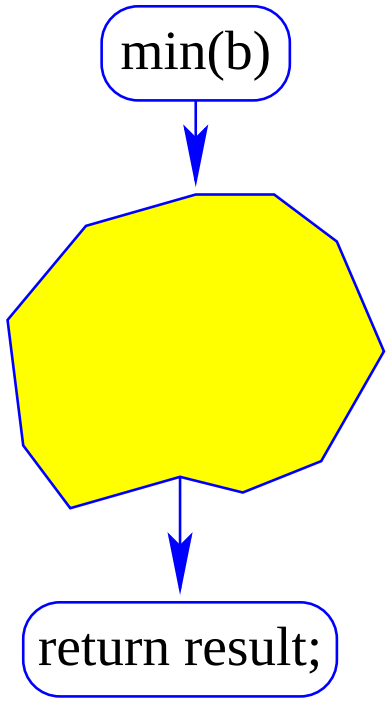
Endknoten

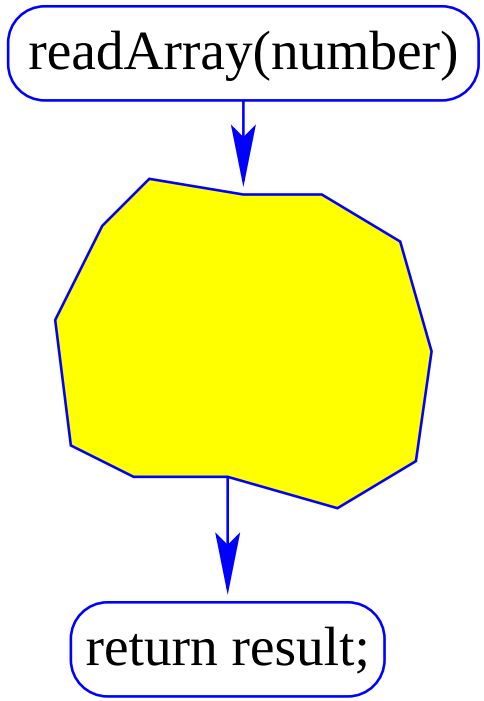
- Für jede Funktion wird ein eigenes Teildiagramm erstellt.
- Ein Aufrufknoten repäsentiert eine Teilberechnung der aufgerufenen Funktion.

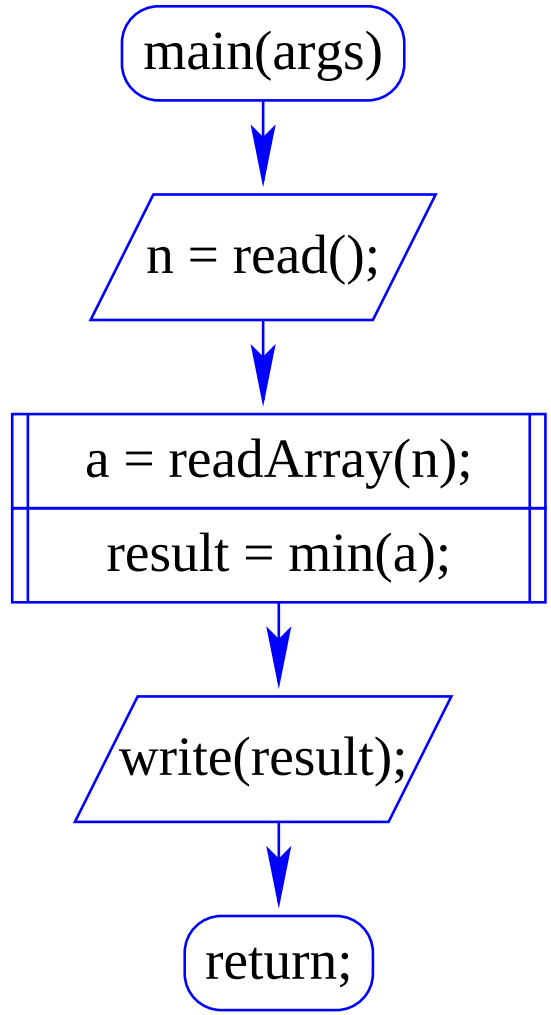
Teildiagramm für die Funktion `min()`:

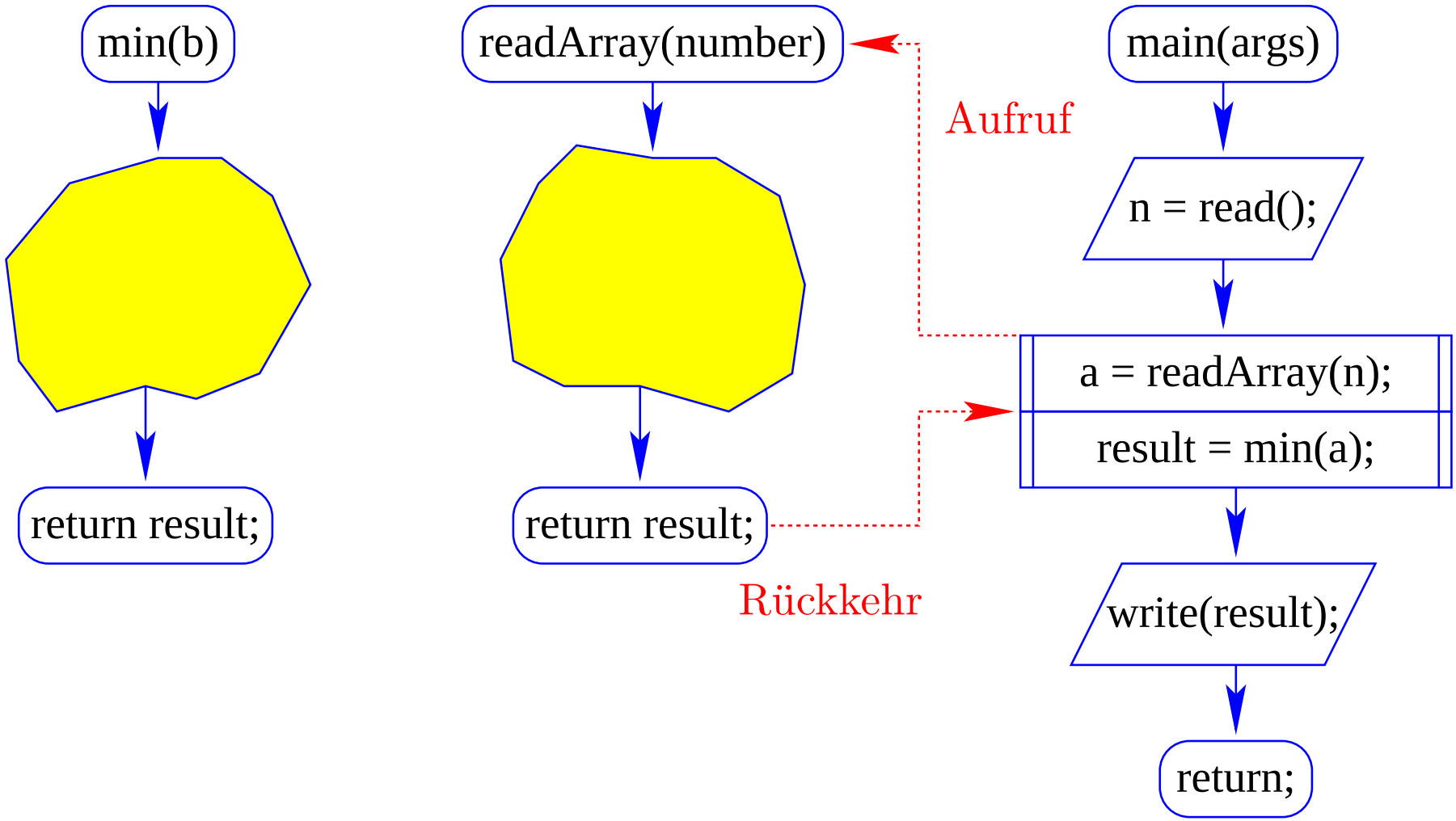


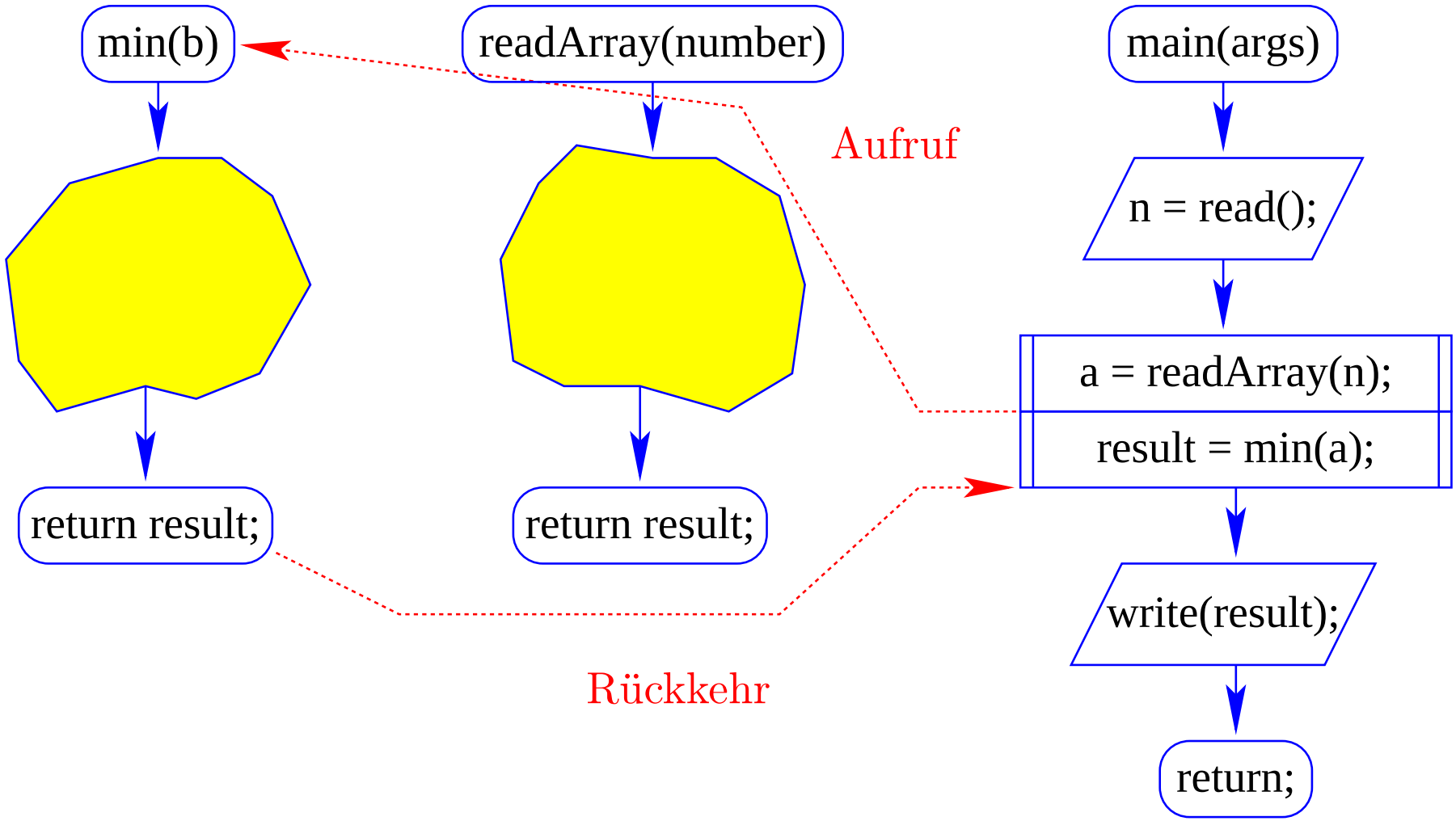
Insgesamt erhalten wir:








min(b)
return result;
readArray(number)
return result;
Aufruf
Rückkehr
main(args)
n = read();
a = readArray(n);
result = min(a);
write(result);
return;


min(b)
readArray(number)
main(args)
Aufruf
n = read();
a = readArray(n);
result = min(a);
return result;
return result;
write(result);
Rückkehr
return;

6 Eine erste Anwendung: Sortieren

Gegeben: eine Folge von ganzen Zahlen.

Gesucht: die zugehörige aufsteigend sortierte Folge.

6 Eine erste Anwendung: Sortieren

Gegeben: eine Folge von ganzen Zahlen.

Gesucht: die zugehörige aufsteigend sortierte Folge.

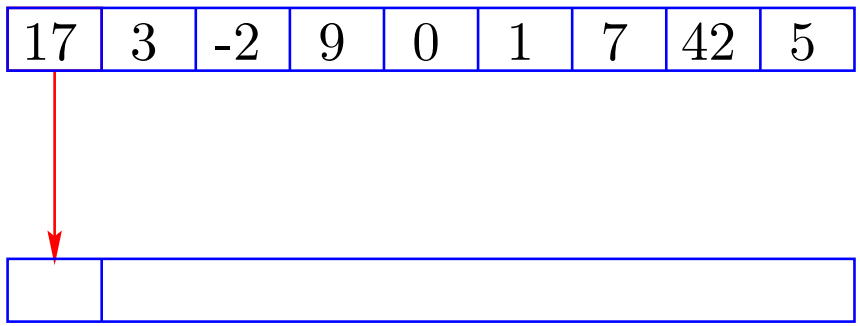
Idee:

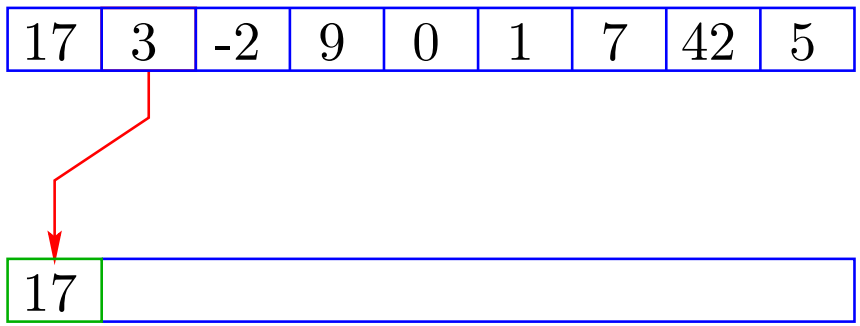
- speichere die Folge in einem Feld ab;
- lege ein weiteres Feld an;
- füge der Reihe nach jedes Element des ersten Felds an der richtigen Stelle in das zweite Feld ein!

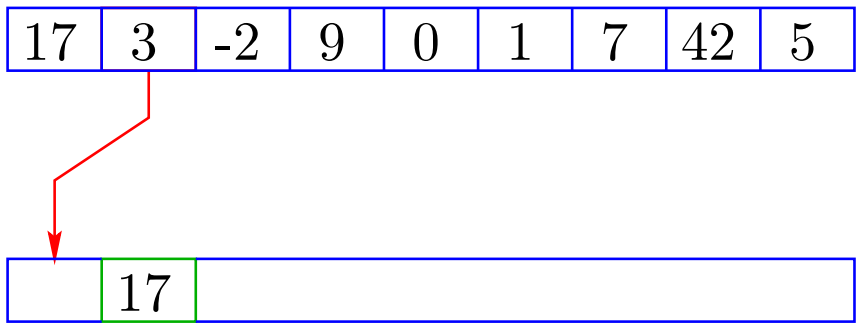
$\Longrightarrow$ Sortieren durch Einfügen ...

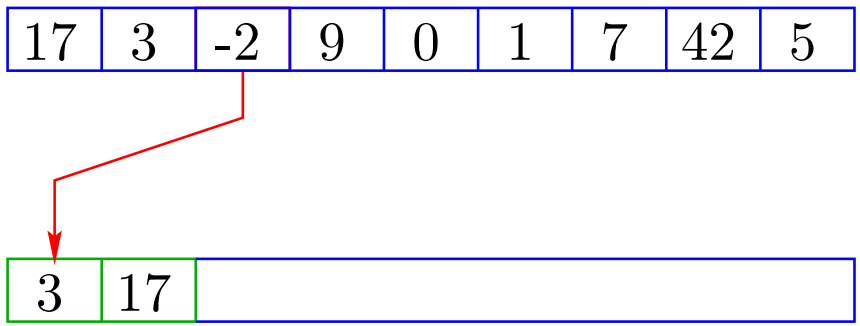
```
public static int[] sort (int[] a) {
   int n = a.length;
   int[] b = new int[n];
   for (int i = 0; i < n; ++i)
      insert (b, a[i], i);
         // b    = Feld, in das eingefügt wird
         // a[i] = einzufügendes Element
         // i    = Anzahl von Elementen in b
   return b;
   } // end of sort ()
```

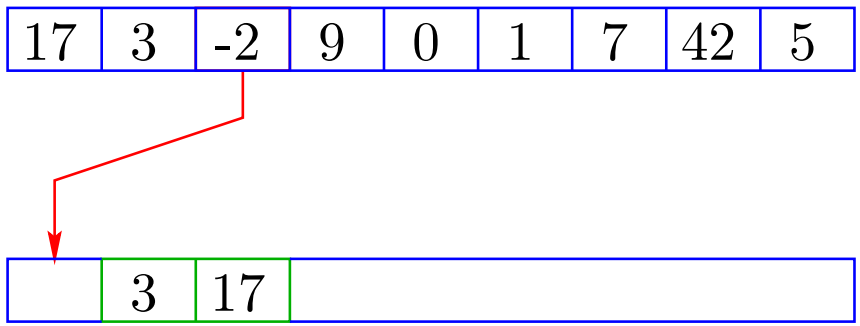
Teilproblem: Wie fügt man ein ???

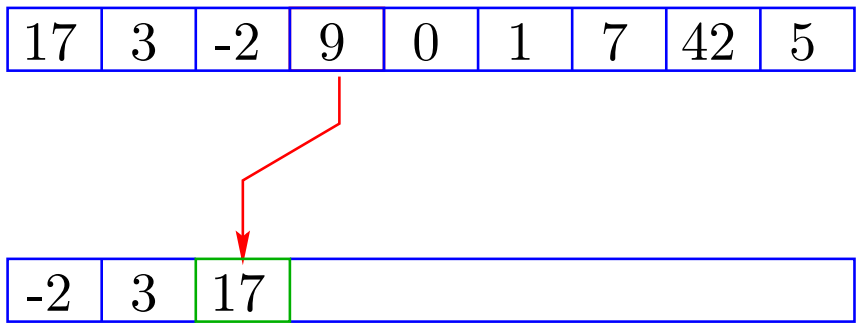

17
3
-2
9
0
1
7
42
5

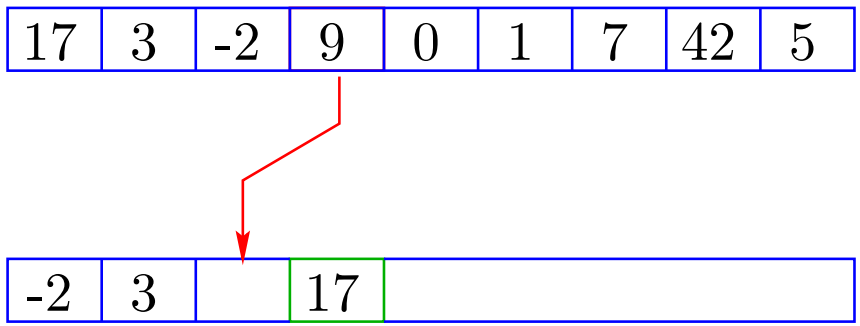

17
3
-2
9
0
1
7
42
5
17

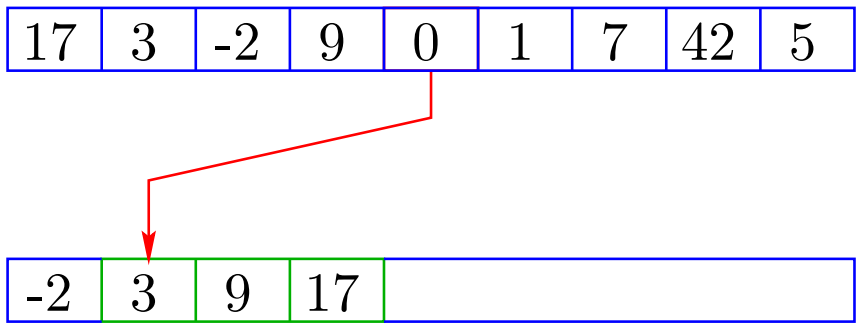

17
3
-2
9
0
1
7
42
5
17


17
3
-2
9
0
1
7
42
5
3
17


17
3
-2
9
0
1
7
42
5
3
17


17
3
-2
9
0
1
7
42
5
-2
3
17


17
3
-2
9
0
1
7
42
5
-2
3
17


17
3
-2
9
0
1
7
42
5
-2
3
9
17