Achtung:

- Die regulären Ausdrücke auf den rechten Regelseiten können sowohl Terminale wie Nicht-Terminale enthalten.
- Deshalb sind kontextfreie Grammatiken mächtiger als reguläre Ausdrücke.

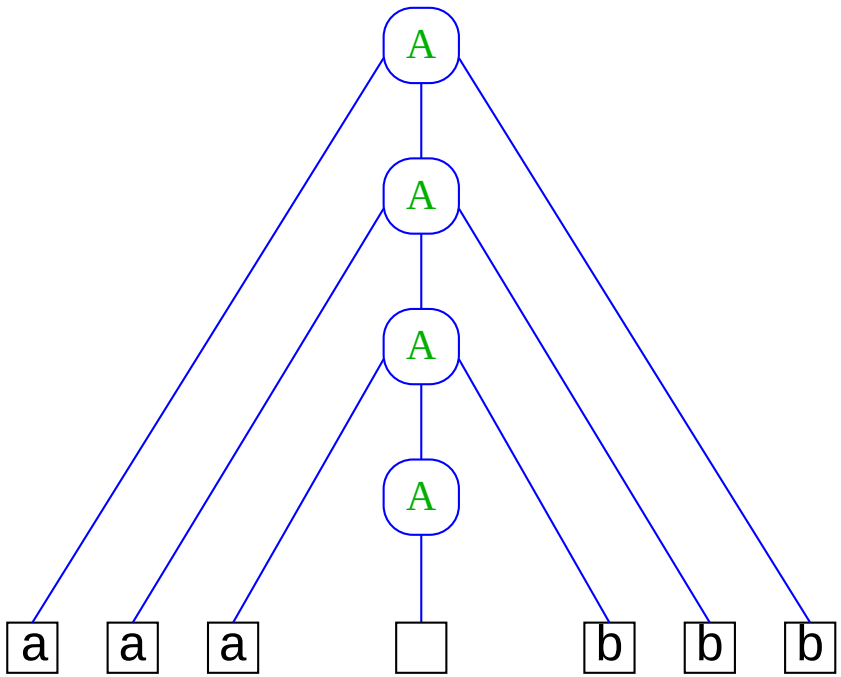
Beispiel:

$$\mathcal{L} = \{\epsilon, \texttt{ab}, \texttt{aabb}, \texttt{aaabbb}, \ldots\}$$

lässt sich mithilfe einer Grammatik beschreiben:

$$\mathrm{A} \quad ::= \quad (\ \texttt{a}\ \mathrm{A}\ \texttt{b}\ )?$$

Syntax-Baum für das Wort `aaabbb` :



Für $\mathcal{L}$ gibt es aber keinen regulären Ausdruck!!!
(↑Automatentheorie)

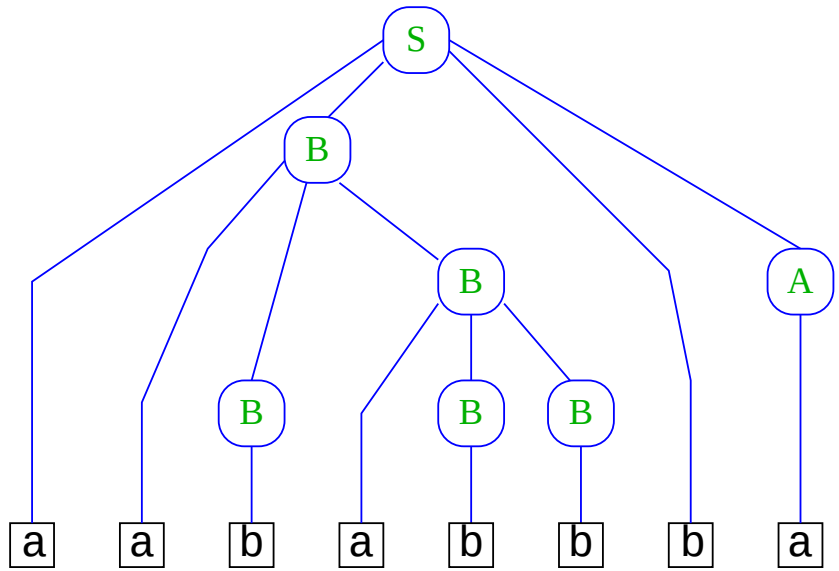
Weiteres Beispiel:

$\mathcal{L}$ = alle Worte mit gleich vielen a's und b's

Zugehörige Grammatik:

$$
\begin{array}{lcl}
\mathrm{S} & ::= & (\ \mathtt{b}\ \mathrm{A} \mid \mathtt{a}\ \mathrm{B}\ )^* \\
\mathrm{A} & ::= & (\ \mathtt{b}\ \mathrm{A}\ \mathrm{A} \mid \mathtt{a}\ ) \\
\mathrm{B} & ::= & (\ \mathtt{a}\ \mathrm{B}\ \mathrm{B} \mid \mathtt{b}\ )
\end{array}
$$

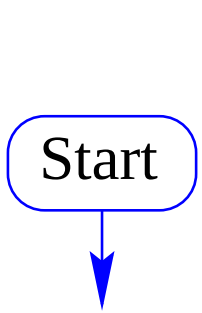
Syntax-Baum für das Wort `aababbba` :



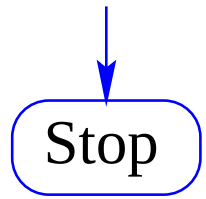
4 Kontrollfluss-Diagramme

In welcher Weise die Operationen eines Programms nacheinander ausgeführt werden, läßt sich anschaulich mithilfe von Kontrollfluss-Diagrammen darstellen.

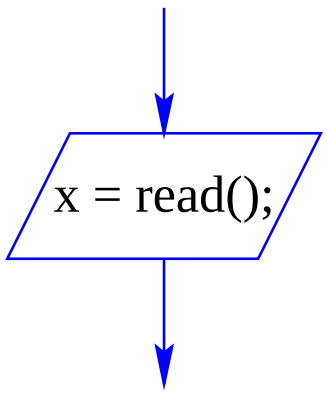
Ingredienzien:



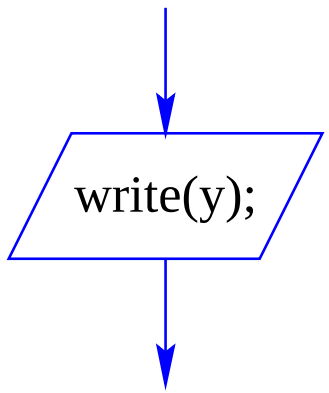
Startknoten



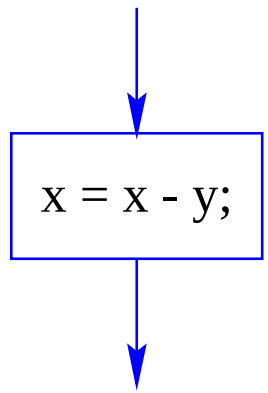
Endknoten



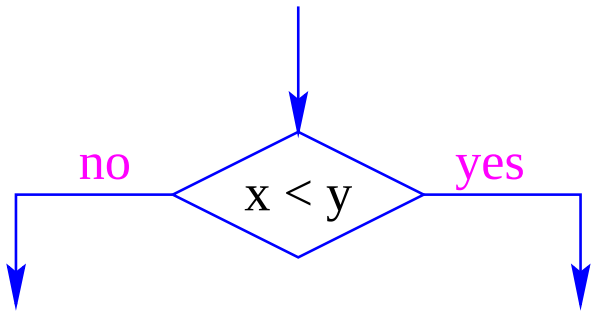
Eingabe



Ausgabe

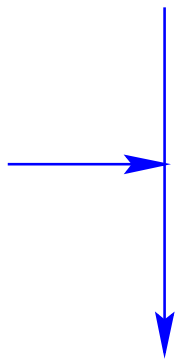


Zuweisung



bedingte Verzweigung

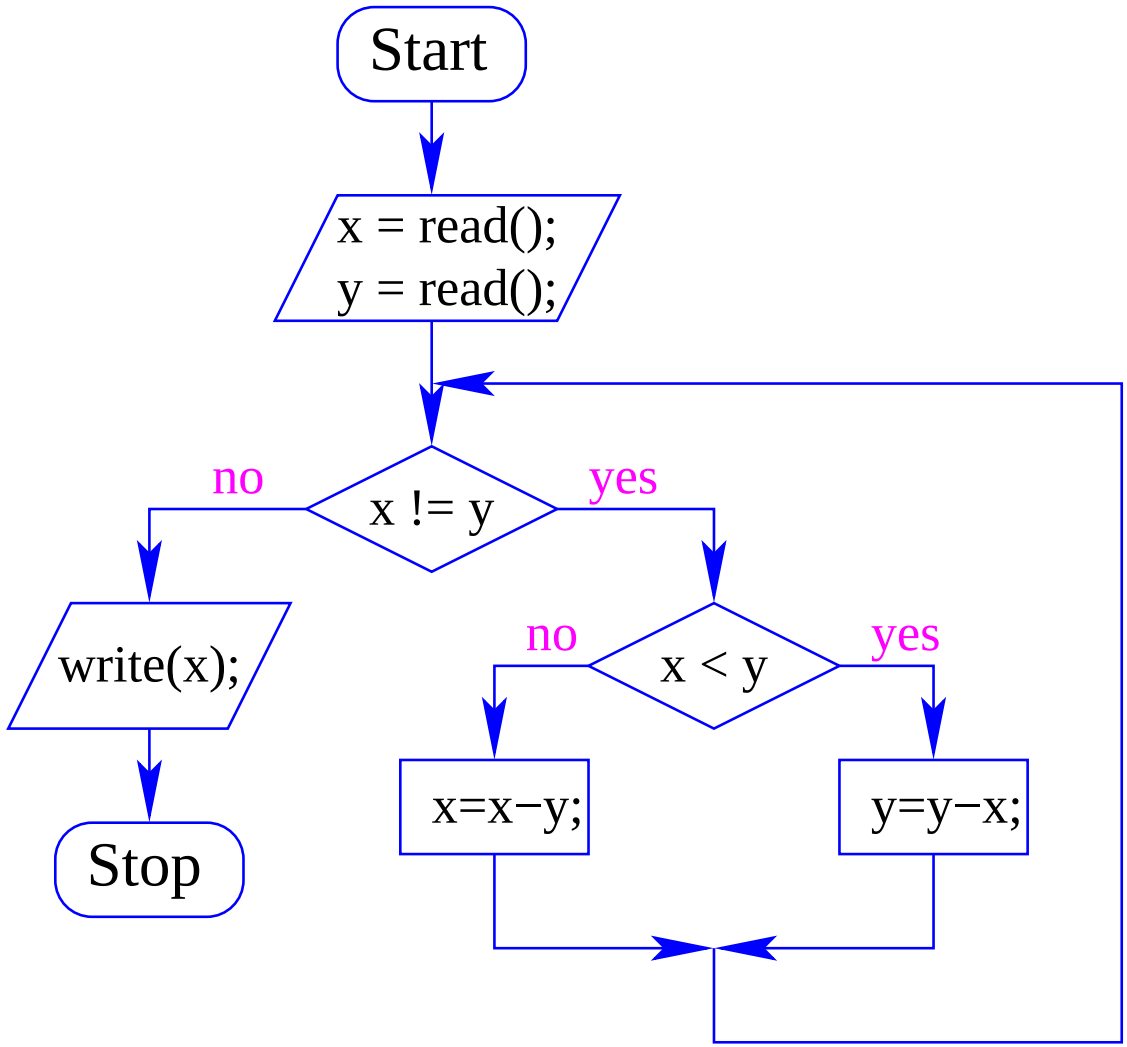
Kante



Zusammenlauf

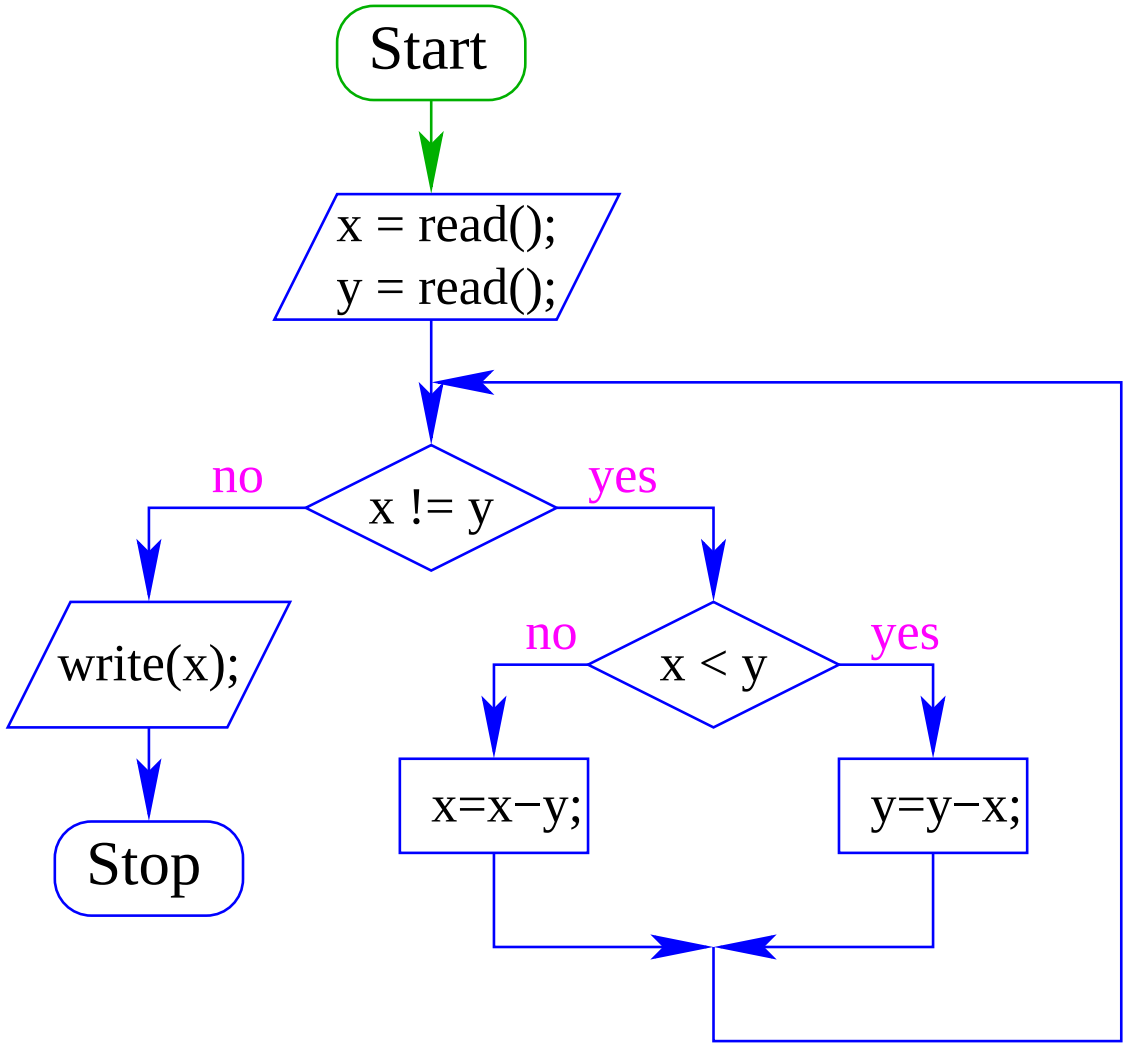
Beispiel:

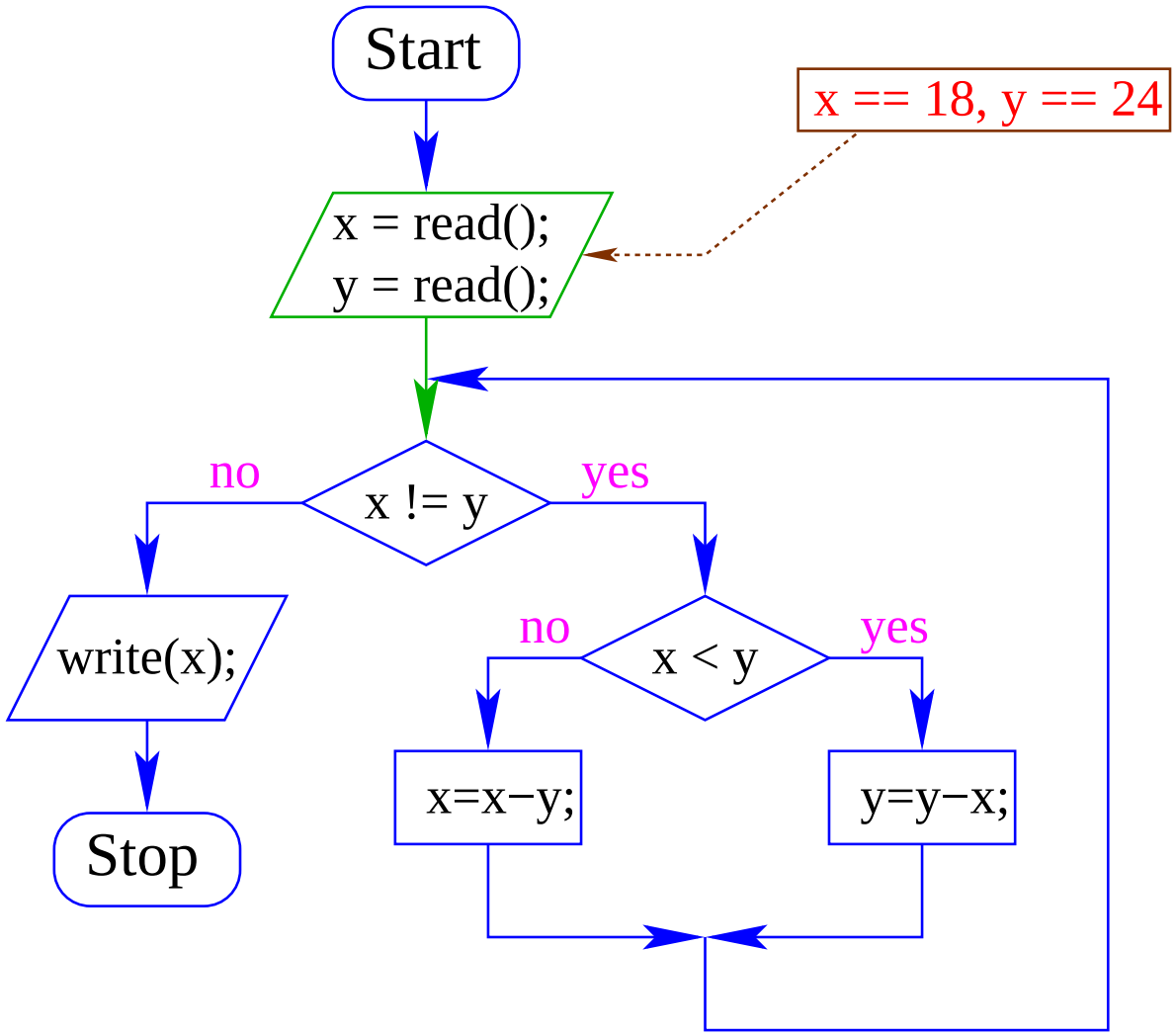
```
int x, y;
x = read();
y = read();
while (x != y)
    if (x < y)
        y = y - x;
    else
        x = x - y;
write(x);
```

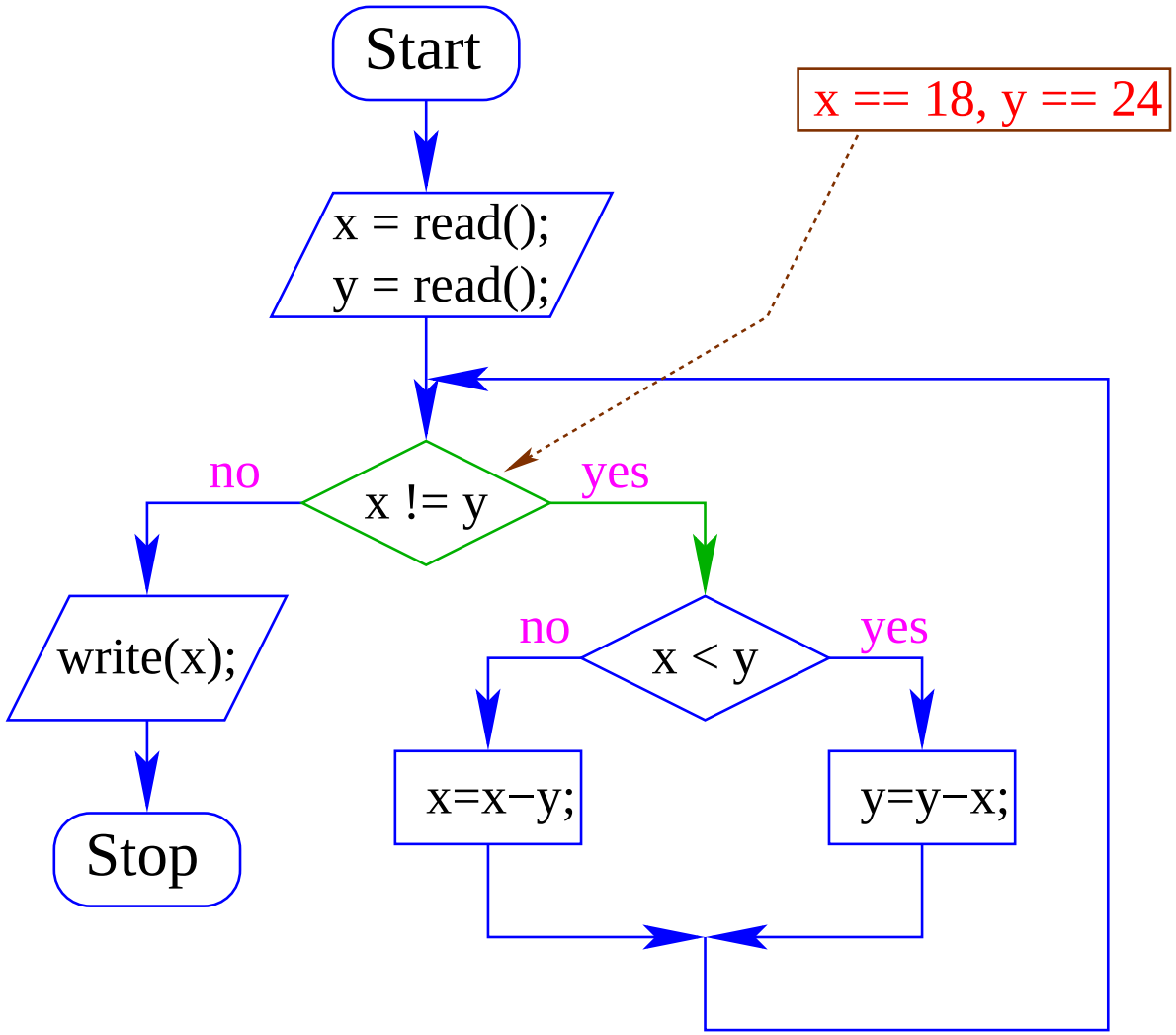


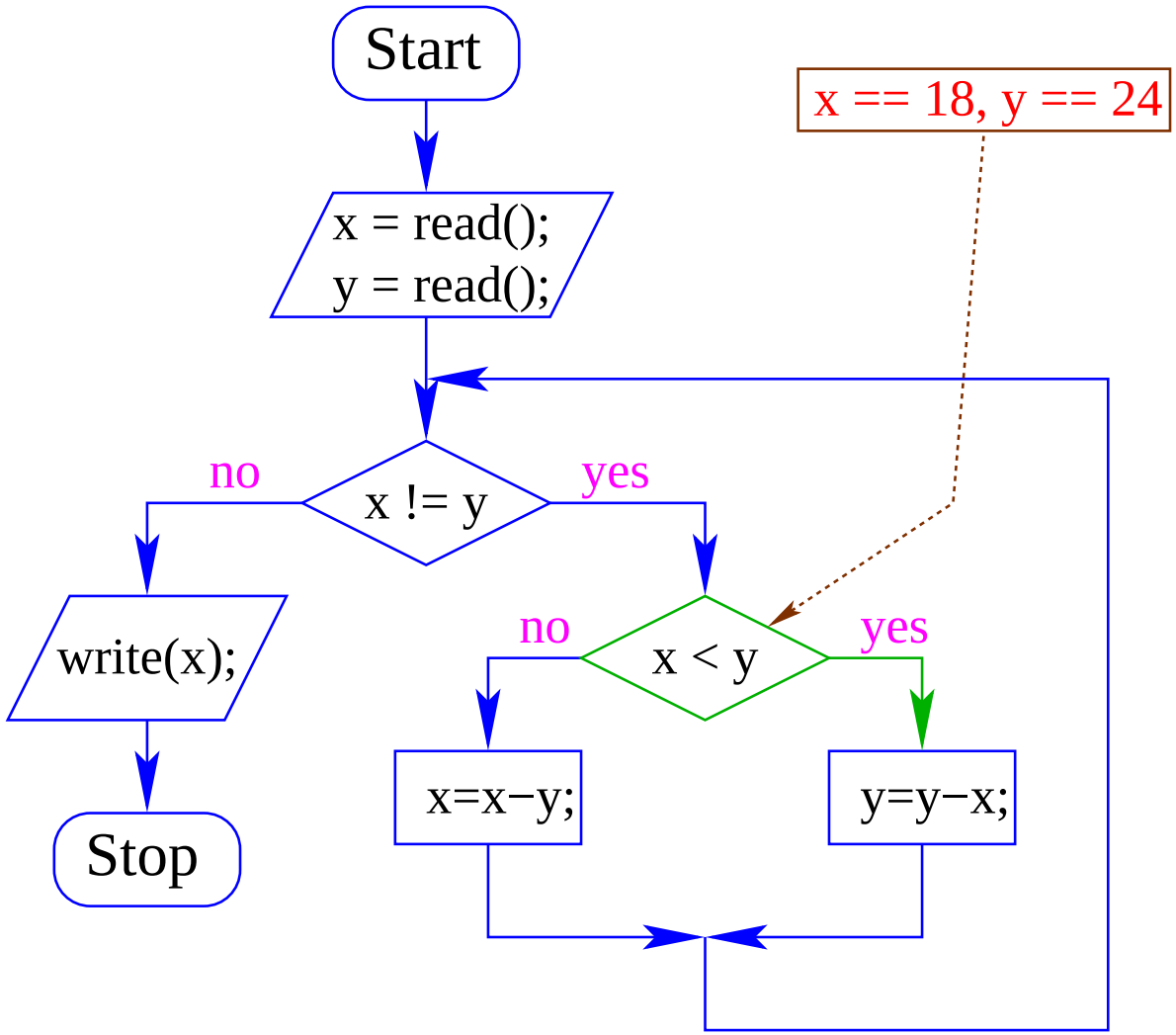
- Die Ausführung des Programms entspricht einem Pfad durch das Kontrollfluss-Diagramm vom Startknoten zum Endknoten.
- Die Deklarationen von Variablen muss man sich am Startknoten vorstellen.
- Die auf dem Pfad liegenden Knoten (außer dem Start- und Endknoten) sind die dabei auszuführenden Operationen bzw. auszuwertenden Bedingungen.
- Um den Nachfolger an einem Verzweigungsknoten zu bestimmen, muss die Bedingung für die aktuellen Werte der Variablen ausgewertet werden.

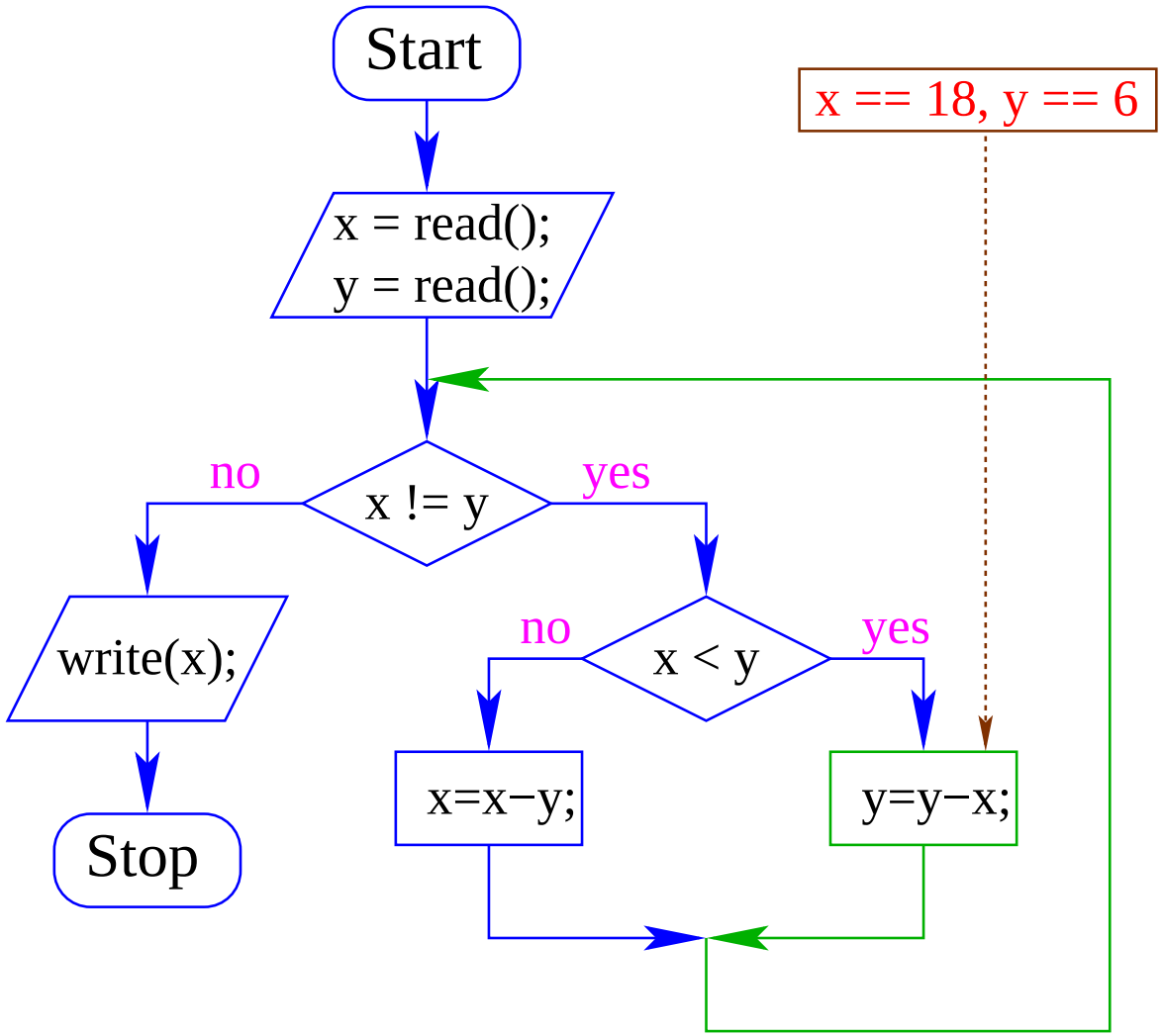
$\Longrightarrow$ operationelle Semantik

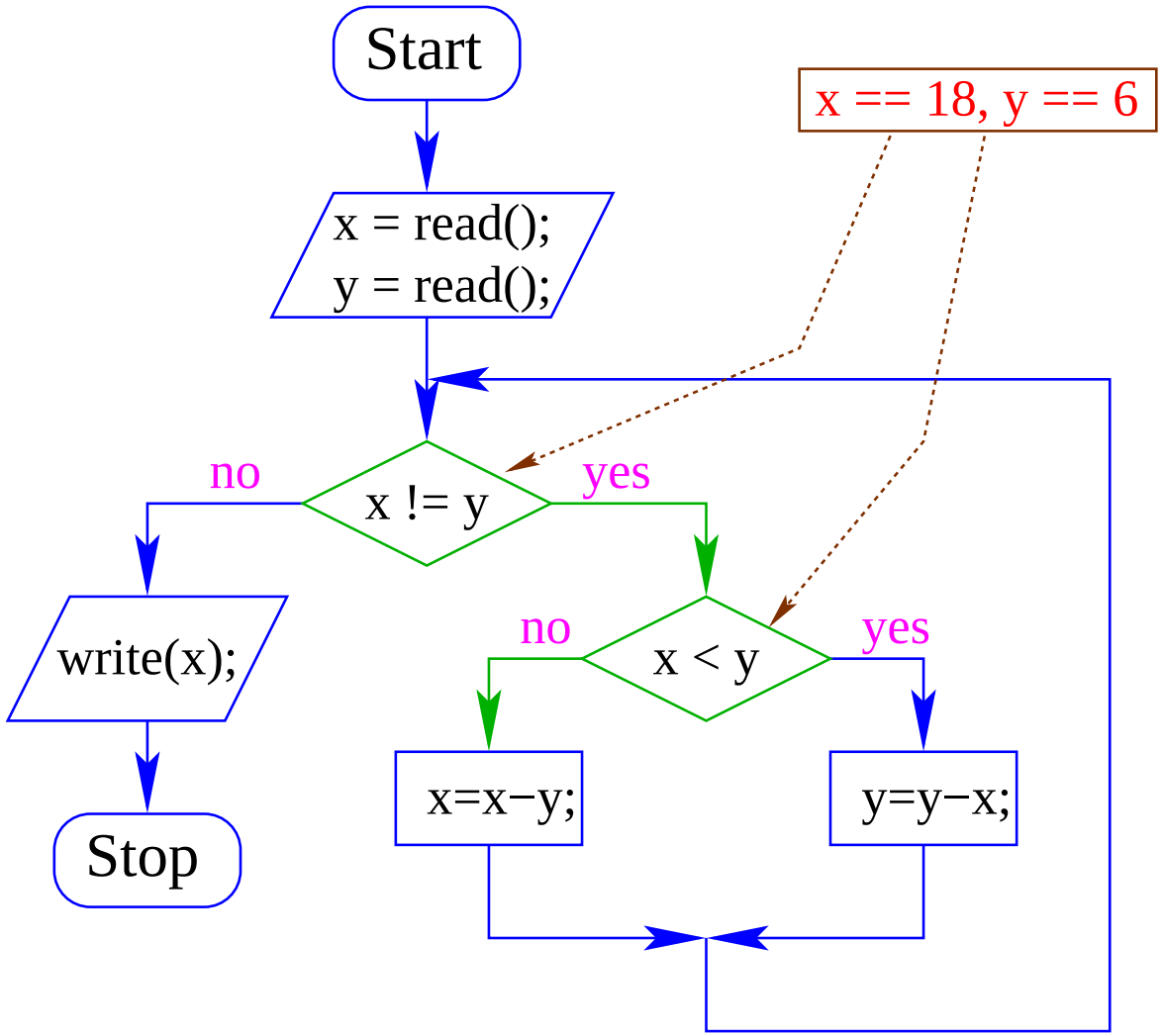

Start
x = read();
y = read();
x != y
no
yes
write(x);
Stop
x < y
no
yes
x=x−y;
y=y−x;

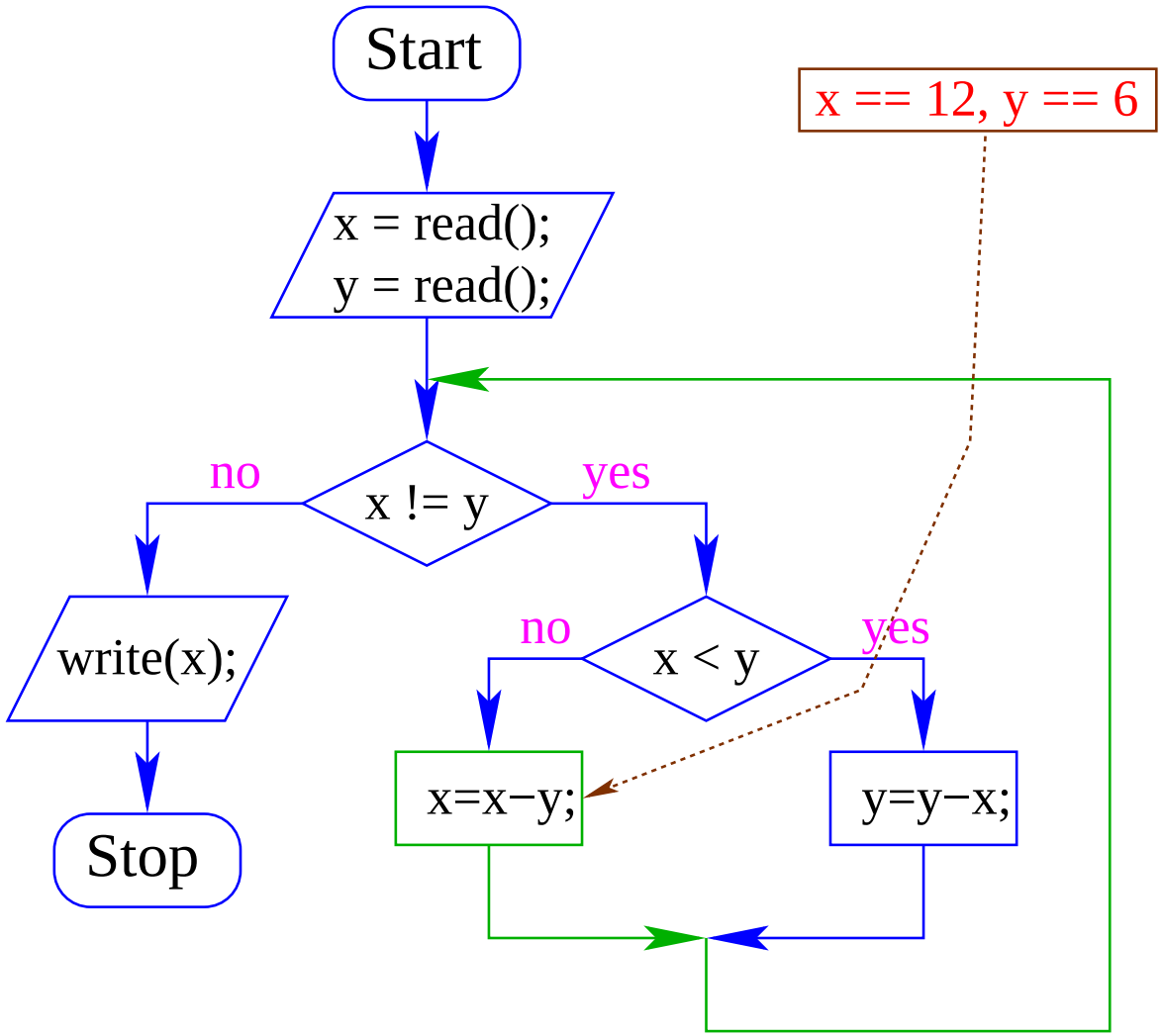

Start
x == 18, y == 24
x = read();
y = read();
no
x != y
yes
write(x);
no
x < y
yes
x=x−y;
y=y−x;
Stop

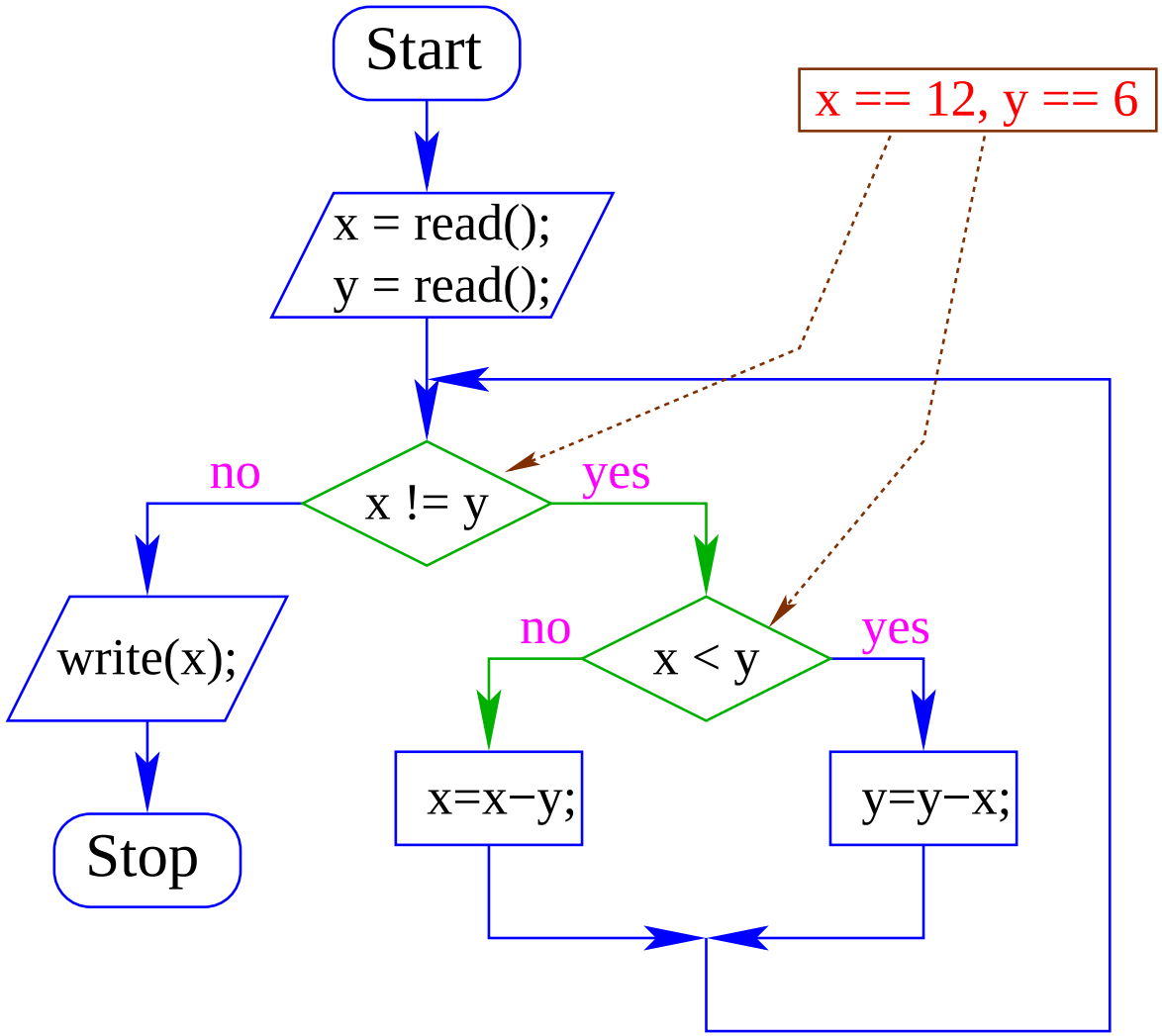

Start
x = read();
y = read();
x == 18, y == 24
x != y
no
yes
write(x);
Stop
x < y
no
yes
x=x−y;
y=y−x;

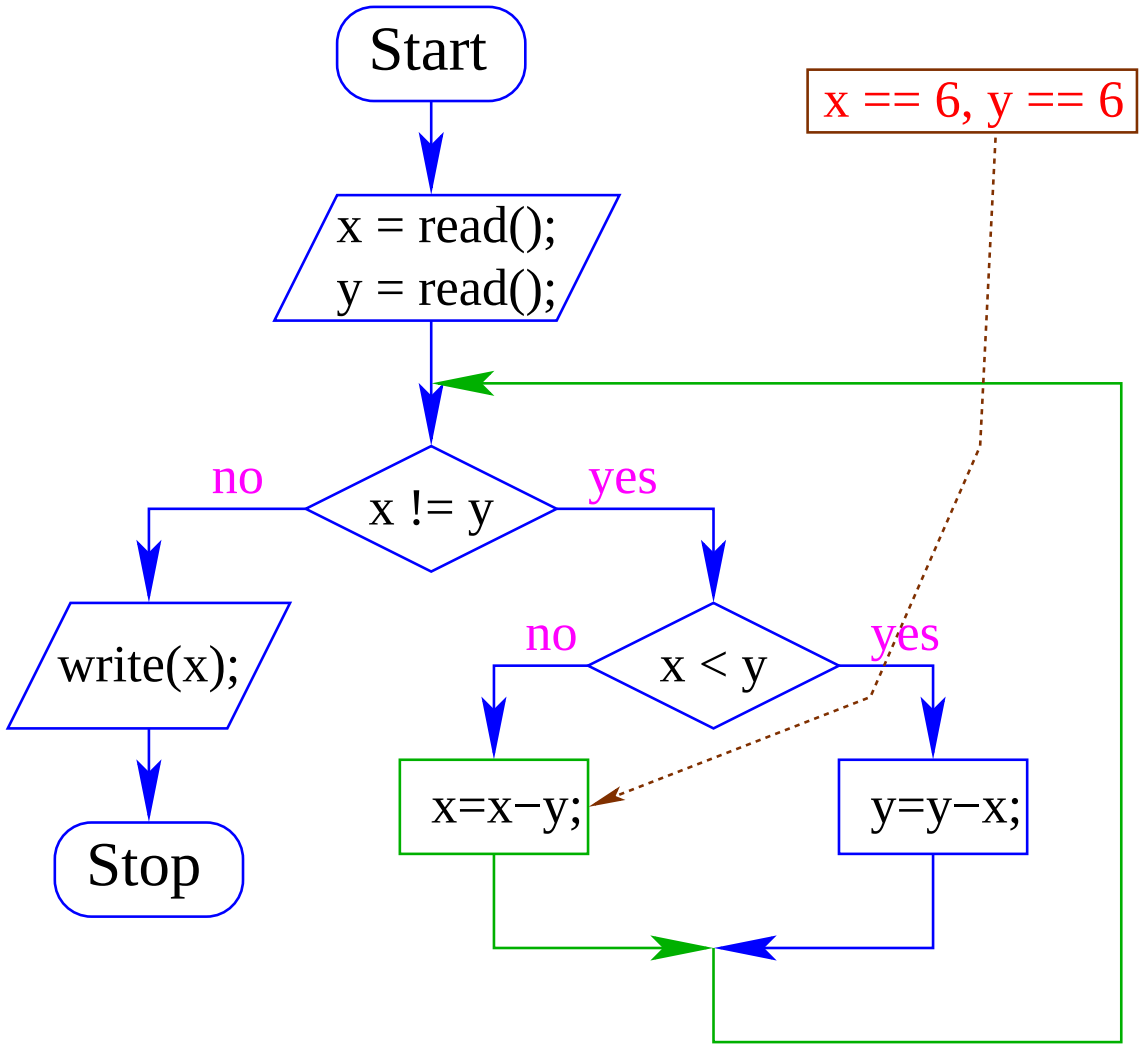

Start
x == 18, y == 24
x = read();
y = read();
no
x != y
yes
write(x);
no
x < y
yes
x=x−y;
y=y−x;
Stop

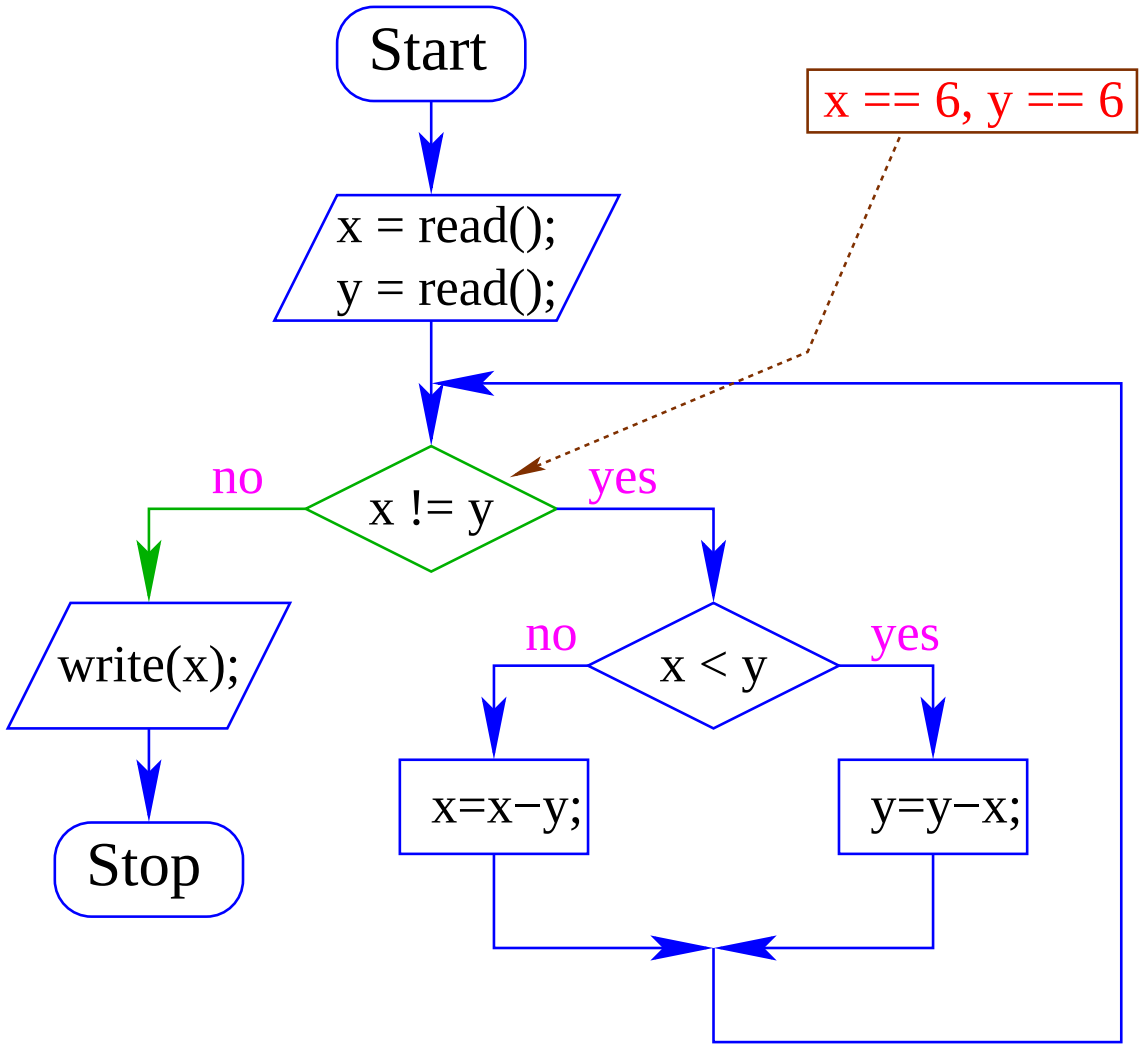

Start
x == 18, y == 6
x = read();
y = read();
no
x != y
yes
write(x);
no
x < y
yes
x=x−y;
y=y−x;
Stop

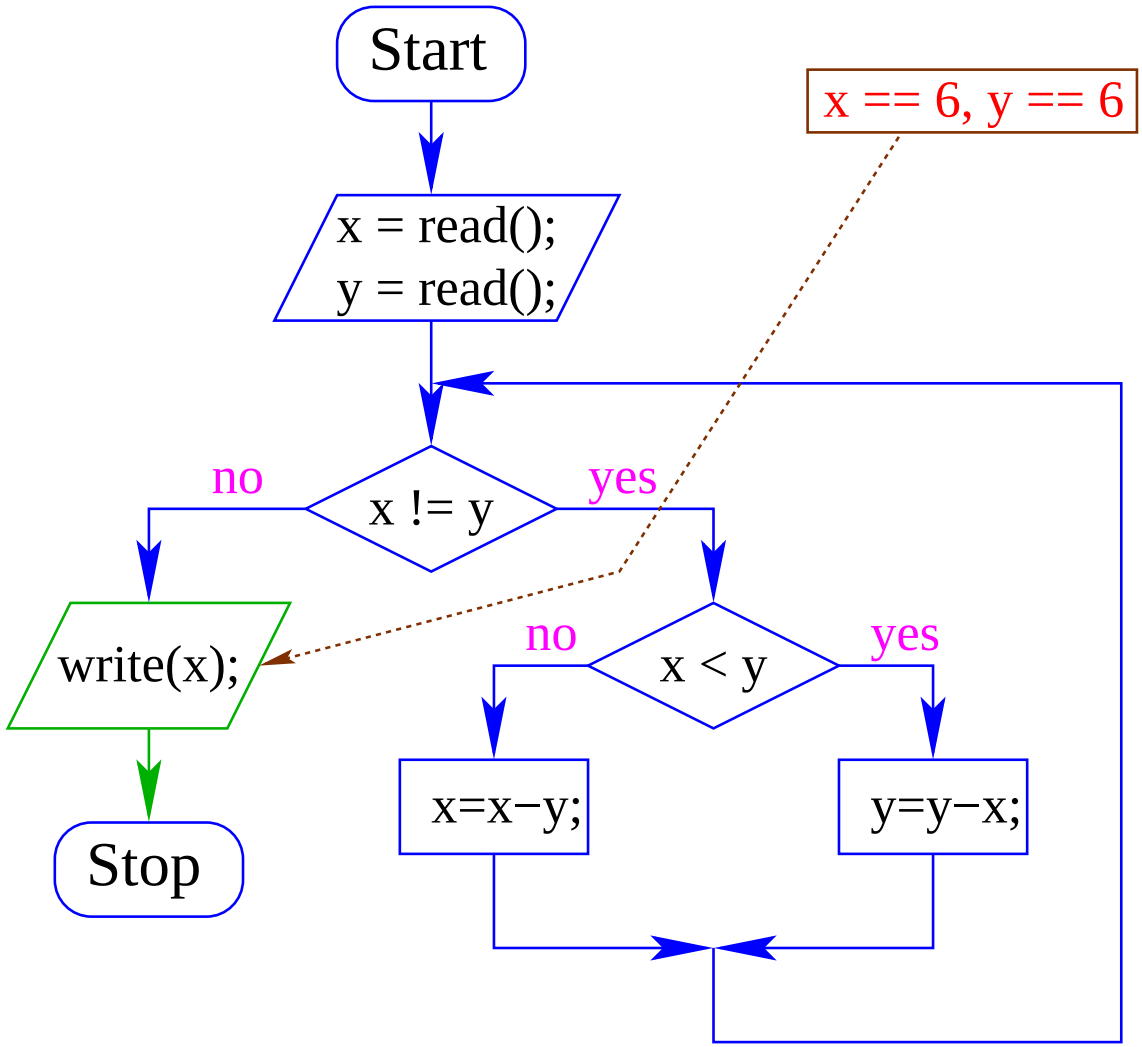

Start
x == 18, y == 6
x = read();
y = read();
no
x != y
yes
write(x);
no
x < y
yes
x=x−y;
y=y−x;
Stop

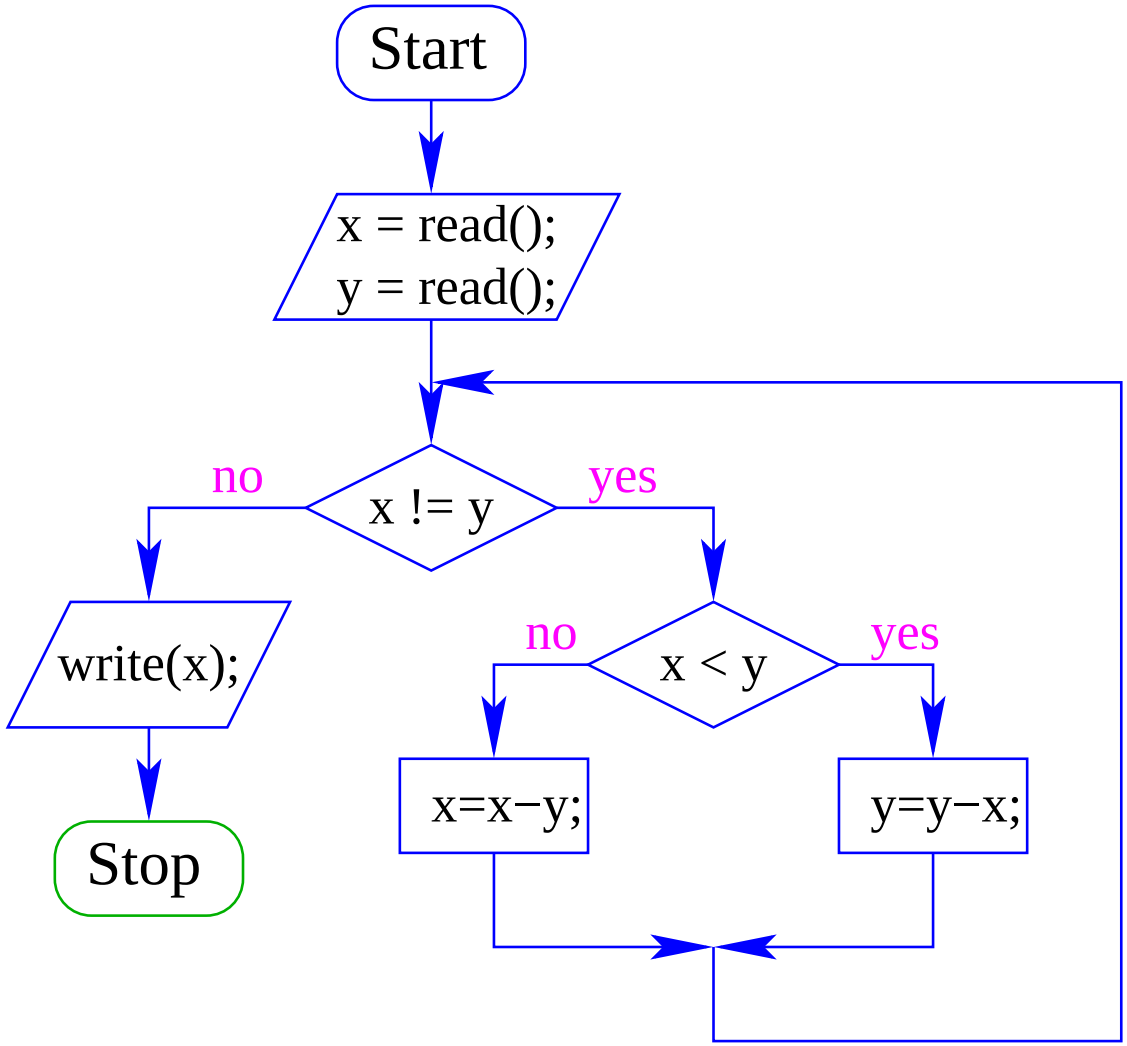

Start
x = read();
y = read();
x == 12, y == 6
x != y
no
yes
write(x);
Stop
x < y
no
yes
x=x−y;
y=y−x;


Start
x = read();
y = read();
x == 12, y == 6
x != y
no
yes
write(x);
Stop
x < y
no
yes
x=x−y;
y=y−x;


Start
x = read();
y = read();
x == 6, y == 6
x != y
no
yes
write(x);
Stop
x < y
no
yes
x=x−y;
y=y−x;


Start
x = read();
y = read();
x == 6, y == 6
x != y
no
yes
write(x);
Stop
x < y
no
yes
x=x−y;
y=y−x;

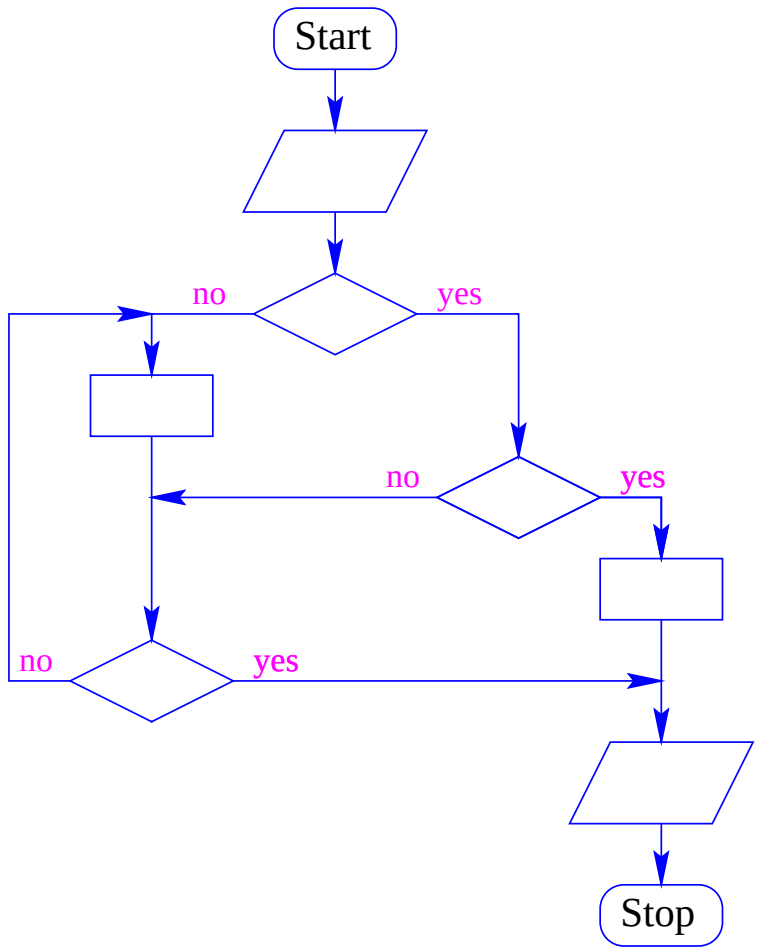

Start
x = read();
y = read();
x == 6, y == 6
x != y
no
yes
write(x);
Stop
x < y
no
yes
x=x−y;
y=y−x;


Start
x = read();
y = read();
x != y
no
yes
write(x);
Stop
x < y
no
yes
x=x−y;
y=y−x;

Achtung:

- Zu jedem MiniJava-Programm lässt sich ein Kontrollfluss-Diagramm konstruieren :-)
- die umgekehrte Richtung gilt zwar ebenfalls, liegt aber nicht so auf der Hand.

Beispiel:



5 Mehr Java

Um komfortabel programmieren zu können, brauchen wir

- mehr Datenstrukturen;
- mehr Kontrollstrukturen :-)

5.1 Mehr Basistypen

- Außer `int`, stellt Java weitere Basistypen zur Verfügung.
- Zu jedem Basistyp gibt es eine Menge möglicher Werte.
- Jeder Wert eines Basistyps benötigt die gleiche Menge Platz, um ihn im Rechner zu repräsentieren.
- Der Platz wird in Bit gemessen.

  (Wie viele Werte kann man mit n Bit darstellen?)

Es gibt vier Sorten ganzer Zahlen:

Typ	Platz	kleinster Wert	größter Wert
`byte`	8	−128	127
`short`	16	−32 768	32 767
`int`	32	−2 147 483 648	2 147 483 647
`long`	64	−9 223 372 036 854 775 808	9 223 372 036 854 775 807

Die Benutzung kleinerer Typen wie `byte` oder `short` spart Platz.

Es gibt vier Sorten ganzer Zahlen:

Typ	Platz	kleinster Wert	größter Wert
`byte`	8	−128	127
`short`	16	−32 768	32 767
`int`	32	−2 147 483 648	2 147 483 647
`long`	64	−9 223 372 036 854 775 808	9 223 372 036 854 775 807

Die Benutzung kleinerer Typen wie `byte` oder `short` spart Platz.

Achtung: Java warnt nicht vor Überlauf/Unterlauf !!

Beispiel:

```
int x = 2147483647; // grösstes int
x = x+1;
write(x);
```

... liefert `-2147483648` ... :-(

- In realem Java kann man bei der Deklaration einer Variablen ihr direkt einen ersten Wert zuweisen (Initialisierung).
- Man kann sie sogar (statt am Anfang des Programms) erst an der Stelle deklarieren, an der man sie das erste Mal braucht!

Es gibt zwei Sorten von Gleitkomma-Zahlen:

Typ	Platz	kleinster Wert	größter Wert	
`float`	32	ca. -3.4e+38	ca. 3.4e+38	7 signifikante Stellen
`double`	64	ca. -1.7e+308	ca. 1.7e+308	15 signifikante Stellen

- Überlauf/Unterlauf liefert die Werte `Infinity` bzw. `-Infinity`.
- Für die Auswahl des geeigneten Typs sollte die gewünschte Genauigkeit des Ergebnisses berücksichtigt werden.
- Gleitkomma-Konstanten im Programm werden als `double` aufgefasst :-)
- Zur Unterscheidung kann man an die Zahl `f` (oder `F`) bzw. `d` (oder `D`) anhängen.

... weitere Basistypen:

Typ	Platz	Werte
`boolean`	1	`true`, `false`
`char`	16	alle Unicode-Zeichen

Unicode ist ein Zeichensatz, der alle irgendwo auf der Welt gängigen Alphabete umfasst, also zum Beispiel:

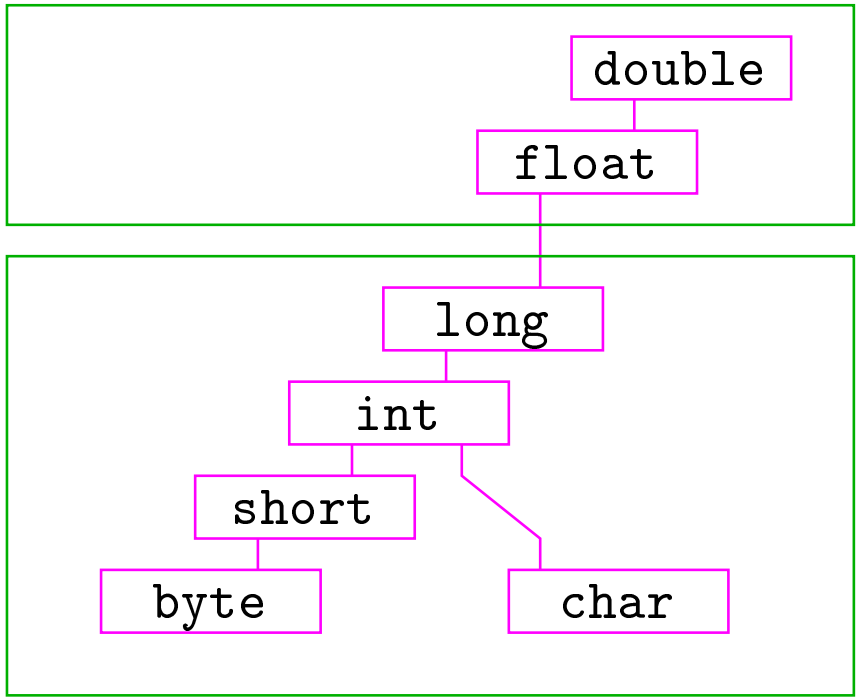
- die Zeichen unserer Tastatur (inklusive Umlaute);
- die chinesischen Schriftzeichen;
- die ägyptischen Hieroglyphen ...

`char`-Konstanten schreibt man mit Hochkommas: `'A'`, `';'`, `'\n'`.

5.2 Mehr über Arithmetik

- Die Operatoren `+`, `-`, `*`, `/` und `%` gibt es für jeden der aufgelisteten Zahltypen :-)

- Werden sie auf ein Paar von Argumenten verschiedenen Typs angewendet, wird automatisch vorher der speziellere in den allgemeineren umgewandelt (impliziter Type Cast) ...



Gleitkomma-Zahlen

ganze Zahlen

Beispiel:

```
short xs = 1;
int x = 999999999;
write(x + xs);
```

... liefert den `int`-Wert `1000000000` ... :-)

```
float xs = 1.0f;
int x = 999999999;
write(x + xs);
```

... liefert den `float`-Wert `1.0E9` ... :-)

Beispiel:

```
short xs = 1;
int x = 999999999;
write(x + xs);
```

... liefert den `int`-Wert `1000000000` ... :-)

```
float xs = 1.0f;
int x = 999999999;
write(x + xs);
```

... liefert den `float`-Wert `1.0E9` ... :-)

... vorausgesetzt, `write()` kann Gleitkomma-Zahlen ausgeben :-)

Achtung:

- Das Ergebnis einer Operation auf `float` kann aus dem Bereich von `float` herausführen. Dann ergibt sich der Wert `Infinity` oder `-Infinity`.

  Das gleiche gilt für `double`.
- Das Ergebnis einer Operation auf Basistypen, die in `int` enthalten sind (außer `char`), liefern ein `int`).
- Wird das Ergebnis einer Variablen zugewiesen, sollte deren Typ dies zulassen :-)
- Mithilfe von expliziten Type Casts lässt sich das (evt. unter Verlust von Information) stets bewerkstelligen.

Beispiele:

`(float) 1.7e+308`	liefert	`Infinity`
`(long) 1.7e+308`	liefert	9223372036854775807
		(d.h. den größten `long`-Wert)
`(int) 1.7e+308`	liefert	2147483647
		(d.h. den größten `int`-Wert)
`(short) 1.7e+308`	liefert	-1
`(int) 1.0e9`	liefert	1000000000
`(int) 1.11`	liefert	1
`(int) -2.11`	liefert	-2

5.3 Strings

Der Datentyp `String` für Wörter ist kein Basistyp, sondern eine Klasse (dazu kommen wir später :-)

Hier behandeln wir nur drei Eigenschaften:

- Werte vom Typ `String` haben die Form `"Hello World!";`
- Man kann Wörter in Variablen vom Typ `String` abspeichern.
- Man kann Wörter mithilfe des Operators "`+`" konkatenieren.

Beispiel:

```
String s0 = "";
String s1 = "Hel";
String s2 = "lo Wo";
String s3 = "rld!";
write(s0 + s1 + s2 + s3);
```

... schreibt `Hello World!` auf die Ausgabe :-)

Beachte:

- Jeder Wert in Java hat eine Darstellung als `String`.
- Wird der Operator "+" auf einen Wert vom Typ `String` und einen anderen Wert x angewendet, wird x automatisch in seine `String`-Darstellung konvertiert ...

$\Longrightarrow$ ... liefert einfache Methode, um `float` oder `double` auszugeben !!!

Beispiel:

```
double x = -0.55e13;
write("Eine Gleitkomma-Zahl: "+x);
```

... schreibt `Eine Gleitkomma-Zahl: -0.55E13` auf die Ausgabe :-)