

Selektion (bedingte Auswahl):

```
int x, y, result;  
x = read();  
y = read();  
if (x > y)  
    result = x - y;  
else  
    result = y - x;  
write(result);
```

- Zuerst wird die Bedingung ausgewertet.
- Ist sie erfüllt, wird die nächste Operation ausgeführt.
- Ist sie nicht erfüllt, wird die Operation nach dem `else` ausgeführt.

Beachte:

- Statt aus einzelnen Operationen können die Alternativen auch aus Statements bestehen:

```
int x;  
x = read();  
if (x == 0)  
    write(0);  
else if (x < 0)  
    write(-1);  
else  
    write(+1);
```

- ... oder aus (geklammerten) Folgen von Operationen und Statements:

```
int x, y;  
x = read();  
if (x != 0) {  
    y = read();  
    if (x > y)  
        write(x);  
    else  
        write(y);  
} else  
    write(0);
```

- ... eventuell fehlt auch der else-Teil:

```
int x, y;  
x = read();  
if (x != 0) {  
    y = read();  
    if (x > y)  
        write(x);  
    else  
        write(y);  
}
```

Auch mit Sequenz und Selektion kann noch nicht viel berechnet werden ... :-)

Iteration (wiederholte Ausführung):

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

- Zuerst wird die Bedingung ausgewertet.
- Ist sie erfüllt, wird der **Rumpf** des `while`-Statements ausgeführt.
- Nach Ausführung des Rumpfs wird das gesamte `while`-Statement erneut ausgeführt.
- Ist die Bedingung nicht erfüllt, fährt die Programm-Ausführung hinter dem `while`-Statement fort.

Jede (partielle) Funktion auf ganzen Zahlen, die überhaupt berechenbar ist, läßt sich mit Selektion, Sequenz, Iteration, d.h. mithilfe eines MiniJava-Programms berechnen :-)

Beweis: ↑ Berechenbarkeitstheorie.

Idee:

Eine Turing-Maschine kann alles berechnen...

Versuche, eine Turing-Maschine zu simulieren!

MiniJava-Programme sind ausführbares Java.

Man muss sie nur geeignet dekorieren :-)

MiniJava-Programme sind ausführbares Java.

Man muss sie nur geeignet dekorieren :-)

Beispiel: Das GGT-Programm

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

Daraus wird das Java-Programm:

```
public class GGT extends MiniJava {
    public static void main (String[] args) {

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

    } // Ende der Definition von main();
} // Ende der Definition der Klasse GGT;
```

Erläuterungen:

- Jedes Programm hat einen **Namen** (hier: GGT).
- Der Name steht hinter dem Schlüsselwort `class` (was eine Klasse, was `public` ist, lernen wir später ... :-)
- Der Datei-Name muss zum Namen des Programms “passen”, d.h. in diesem Fall `GGT.java` heißen.
- Das **MiniJava**-Programm ist der Rumpf des **Hauptprogramms**, d.h. der Funktion `main()`.
- Die Programm-Ausführung eines **Java**-Programms startet stets mit einem Aufruf von dieser Funktion `main()`.
- Die Operationen `write()` und `read()` werden in der Klasse `MiniJava` definiert.
- Durch `GGT extends MiniJava` machen wir diese Operationen innerhalb des GGT-Programms verfügbar.

Die Klasse MiniJava ist in der Datei MiniJava.java definiert:

```
import javax.swing.JOptionPane;
import javax.swing.JFrame;
public class MiniJava {
    public static int read () {
        JFrame f = new JFrame ();
        String s = JOptionPane.showInputDialog (f, "Eingabe:");
        int x = 0; f.dispose ();
        if (s == null) System.exit (0);
        try { x = Integer.parseInt (s.trim ());
        } catch (NumberFormatException e) { x = read (); }
        return x;
    }
    public static void write (String x) {
        JFrame f = new JFrame ();
        JOptionPane.showMessageDialog (f, x, "Ausgabe",
            JOptionPane.PLAIN_MESSAGE);
        f.dispose ();
    }
    public static void write (int x) { write (""+x); }
}
```

... weitere Erläuterungen:

- Die Klasse MiniJava werden wir im Lauf der Vorlesung im Detail verstehen lernen :-)
- Jedes Programm sollte **Kommentare** enthalten, damit man sich selbst später noch darin zurecht findet!
- Ein Kommentar in **Java** hat etwa die Form:

```
// Das ist ein Kommentar!!!
```

- Wenn er sich über mehrere Zeilen erstrecken soll, kann er auch so aussehen:

```
/* Dieser Kommentar geht  
"über mehrere Zeilen! */
```

Das Programm GGT kann nun übersetzt und dann ausgeführt werden:

```
seidl> javac GGT.java  
seidl> java GGT
```

- Der Compiler `javac` liest das Programm aus den Dateien `GGT.java` und `MiniJava.java` ein und erzeugt für sie JVM-Code, den er in den Dateien `GGT.class` und `MiniJava.class` ablegt.
- Das Laufzeitsystem `java` liest die Dateien `GGT.class` und `MiniJava.class` ein und führt sie aus.

Achtung:

- **MiniJava** ist sehr primitiv.
- Die Programmiersprache **Java** bietet noch eine Fülle von Hilfsmitteln an, die das Programmieren erleichtern sollen. Insbesondere gibt es
- viele weitere Datenstrukturen (nicht nur int) und
- viele weitere Kontrollstrukturen.

... kommt später in der Vorlesung :-)

3 Syntax von Programmiersprachen

Syntax (“Lehre vom Satzbau”):

- formale Beschreibung des Aufbaus der “Worte” und “Sätze”, die zu einer Sprache gehören;
- im Falle einer **Programmier**-Sprache Festlegung, wie Programme aussehen müssen.

Hilfsmittel bei natürlicher Sprache:

- Wörterbücher;
- Rechtschreibregeln, Trennungsregeln, Grammatikregeln;
- Ausnahme-Listen;
- Sprach-“Gefühl”.

Hilfsmittel bei Programmiersprachen:

- Listen von **Schlüsselwörtern** wie `if`, `int`, `else`, `while` ...
- Regeln, wie einzelne Worte (**Tokens**) z.B. **Namen** gebildet werden.

Frage:

Ist `x10` ein zulässiger Name für eine Variable?
oder `_ab$` oder `A#B` oder `0A?B` ...

- Grammatikregeln, die angeben, wie größere Komponenten aus kleineren aufgebaut werden.

Frage:

Ist ein `while`-Statement im `else`-Teil erlaubt?

- Kontextbedingungen.

Beispiel:

Eine Variable muss erst deklariert sein, bevor sie verwendet wird.

⇒ formalisierter als natürliche Sprache

⇒ besser für maschinelle Verarbeitung geeignet

Semantik (“Lehre von der Bedeutung”):

- Ein Satz einer (natürlichen) Sprache verfügt zusätzlich über eine **Bedeutung**, d.h. teilt einem Hörer/Leser einen Sachverhalt mit (↑**Information**)
- Ein Satz einer Programmiersprache, d.h. ein Programm verfügt ebenfalls über eine **Bedeutung** :-)

Die Bedeutung eines Programms ist

- alle möglichen **Ausführungen** der beschriebenen Berechnung (↑ **operationelle Semantik**); oder
- die definierte **Abbildung** der Eingaben auf die Ausgaben (↑ **denotationelle Semantik**).

Die Bedeutung eines Programms ist

- alle möglichen **Ausführungen** der beschriebenen Berechnung (↑ **operationelle Semantik**); oder
- die definierte **Abbildung** der Eingaben auf die Ausgaben (↑ **denotationelle Semantik**).

Achtung!

Ist ein Programm **syntaktisch korrekt**, heißt das noch lange nicht, dass es auch das “richtige” tut, d.h. **semantisch korrekt** ist !!!

3.1 Reservierte Wörter

- `int`
 - Bezeichner für Basis-Typen;
- `if, else, while`
 - Schlüsselwörter aus Programm-Konstrukten;
- `(,), ", ', {, }, ,, ;`
 - Sonderzeichen.

3.2 Was ist ein erlaubter Name?

Schritt 1: Angabe der erlaubten Zeichen:

letter ::= \$ | _ | a | ... | z | A | ... | Z
digit ::= 0 | ... | 9

3.2 Was ist ein erlaubter Name?

Schritt 1: Angabe der erlaubten Zeichen:

`letter` ::= \$ | _ | a | ... | z | A | ... | Z
`digit` ::= 0 | ... | 9

- `letter` und `digit` bezeichnen **Zeichenklassen**, d.h. Mengen von Zeichen, die gleich behandelt werden.
- Das Symbol “|” trennt zulässige Alternativen.
- Das Symbol “...” repräsentiert die Faulheit, alle Alternativen wirklich aufzuzählen :-)

Schritt 2: Angabe der Anordnung der Zeichen:

`name ::= letter (letter | digit)*`

- Erst kommt ein Zeichen der Klasse `letter`, dann eine (eventuell auch leere) Folge von Zeichen entweder aus `letter` oder aus `digit`.
- Der Operator “*” bedeutet “beliebig ofte Wiederholung” (“weglassen” ist 0-malige Wiederholung :-).
- Der Operator “*” ist ein **Postfix**-Operator. Das heißt, er steht hinter seinem Argument.

Beispiele:

- `_178`
`Das_ist_kein_Name`
`x`
`-`
`$Password$`

... sind legale Namen :-)

- 5ABC
!Hallo!
x'
-178

... sind keine legalen Namen :-)

- 5ABC
!Hallo!
x'
-178

... sind keine legalen Namen :-)

Achtung:

Reservierte Wörter sind als Namen verboten !!!

3.3 Ganze Zahlen

Werte, die direkt im Programm stehen, heißen **Konstanten**.

Ganze Zahlen bestehen aus einem Vorzeichen (das evt. auch fehlt) und einer nichtleeren Folge von Ziffern:

sign ::= + | -

number ::= sign ? digit digit*

3.3 Ganze Zahlen

Werte, die direkt im Programm stehen, heißen **Konstanten**.

Ganze Zahlen bestehen aus einem Vorzeichen (das evt. auch fehlt) und einer nichtleeren Folge von Ziffern:

$$\begin{aligned} \text{sign} & ::= + \mid - \\ \text{number} & ::= \text{sign} ? \text{digit digit}^* \end{aligned}$$

- Der Postfix-Operator “?” besagt, dass das Argument eventuell auch fehlen darf, d.h. einmal oder keinmal vorkommt.
- Wie sähe die Regel aus, wenn wir führende Nullen verbieten wollen?

Beispiele:

- -17
+12490
42
0
-00070

... sind alles legale int-Konstanten.

- "Hello World!"
-0.5e+128

... sind keine int-Konstanten.

Ausdrücke, die aus Zeichen (-klassen) mithilfe von

| (Alternative)

* (Iteration)

(Konkatenation) sowie

? (Option)

... aufgebaut sind, heißen **reguläre Ausdrücke**^a

(↑ **Automatentheorie**).

^aGelegentlich sind auch ϵ , d.h. das "leere Wort" sowie \emptyset , d.h. die leere Menge zugelassen.