Tagesprogramm

Syntax und Semantik

Varianten von Schleifen

Varianten von Programmverzweigungen

Darstellung der Syntax

Syntax: Aufbau der Sätze bzw. Programme

EBNF: Meta-Sprache zur Beschreibung der Syntax formaler Sprachen

Sequenz:	A B	zuerst kommt A, dann kommt B
Auswahl:	A | B	A oder B
Wiederholung:	{ A }	A beliebig oft wiederholt (keinmal, einmal, ...)
Option:	[A]	entweder A oder nichts

Terminalsymbole rot

```
Ausdruck ::= Literal
           | Name [ ( [ Ausdruck { , Ausdruck } ] ) ]
           | Ausdruck ( + | - | * | / ) Ausdruck
           | ...
```

Aufgabe: Syntaktische Korrektheit

```
Expr  ::=  [ - ] Value [ ( + | - ) Value ]
Value ::=  ( Expr )
         |  0
         |  1 { 0 | 1 }
```

Welche der folgenden Ausdrücke entsprechen dieser Grammatik als Expr?

A `00`

B `--0`

C `((0))`

D `10 + 11 - 0`

E `(10 - 11) + (0 - 1)`

F `-1 - -0`

G `((()))`

H `((- 1 + 1))`

Semantik

Bedeutung von Begriffen, Sätzen, Programmen

nicht einfach darstellbar;
daher verwenden wir textuelle Beschreibung bzw. Pseudocode

While-Schleife

Anweisung	::=	While-Schleife
	|	Block
	|	[Ausdruck] **;**
	|	...
While-Schleife	::=	**while (** Ausdruck **)** Anweisung
Block	::=	**{** { Anweisung } **}**

```
while (x <= 3) { x = x + 1; }

while (x <= 3) x = x + 1;

while (x <= 3) x++;

while (x++ <= 3);
```

werte Ausdruck aus;
wenn Ausdruck wahr, dann
    führe Anweisung aus;
    beginne wieder von vorne;
sonst fertig

Do-While-Schleife

Do-While-Schleife ::= **do** Anweisung **while ( ) ;**

führe Anweisung aus (mindestens einmal);
werte Ausdruck aus;
wenn Ausdruck wahr, dann beginne wieder von vorne;
sonst fertig

```
do { x = x + 1; } while (x <= 3);
```

```
do x = x + 1; while (x <= 3);
```

```
x = x + 1; while (x <= 3) { x = x + 1; }
```

For-Schleife

For-Schleife ::= **for** (Initialisierung ; Ausdruck ; Inkrement) Anweisung

for (Initialisierung ; Ausdruck ; Inkrement) Anweisung

ist äquivalent zu

Initialisierung ; **while** (Ausdruck) { Anweisung Inkrement ; }

```
for (int i = 1; i <= 3; i++) { x = x * 2; }
int i = 1; while (i <= 3) { { x = x * 2; } i++; }
int i; for (i = 1; i <= 3; i = i + 1) x = x * 2;
int i = 1; for (; i <= 3 ;) { x = x * 2; i = i + 1; }
```

Break und Continue im Schleifenrumpf

Anweisung	::=	**break ;**	bricht Schleife und Block ab
	|	**continue ;**	bricht Block ab und startet nächste Iteration
	|	...	

nicht verwenden!

```
while(i<=3){if(i!=2){sum=sum+i;} i++;}

while(true){if(i!=2){sum=sum+i;} i++; if(i>3){break;}}

for(; i<=3; i++){if(i!=2){sum=sum+i;}}

for(; i<=3; i++){if(i!=2){continue;} sum=sum+i;}

for(;; i++){if(i!=2){continue;} sum=sum+i; if(i>3){break;}}
```

Aufgabe:
Wann ist welche Schleifenart zu bevorzugen?

Suchen Sie in Gruppen zu 2 oder 3 Personen eine Antwort auf diese Frage.

Zeit: 3 Minuten

If-Anweisung

If-Anweisung ::= **if** **(** Ausdruck **)** Anweisung$_1$ [**else** Anweisung$_2$]

werte Ausdruck aus;
wenn Ausdruck wahr, dann führe Anweisung$_1$ aus;
sonst wenn `else`-Klausel vorhanden, dann führe Anweisung$_2$ aus;

```
if (x <= 3) { x = 4; } else { if (b) { x = 7; }}
if (x <= 3) { x = 4; } else if (b) { x = 7; }
if (x <= 3) x = 4; else if (b) x = 7;
if (x > 3) { if (b) { x = 7; } } else { x = 4; }
if (x > 3) { if (b) x = 7; } else x = 4;
```

Switch-Anweisung

Switch ::= **switch (Ausdruck) { { Case } [default : { Anweisung }] }**

Case ::= **case** Konstante **:** { Anweisung }

werte Ausdruck aus, suche Konstante mit diesem Wert in Konstantenlisten;
wenn gefunden, dann beginne nach entsprechendem `:`,
sonst wenn `default`-Klausel vorhanden, dann beginne nach `default:`,
sonst fertig;
„beginne nach“ bedeutet: Führe alle danach stehenden Anweisungen bis } aus;
wenn `break;` ausgeführt, dann fertig

```
switch(x) {
    case 1: y = 2; break;
    case 7: z = 5; break;
    default: y = 3;
}
```

If versus Switch

```
int note = ...;
String beurt;

if (note == 1)
    beurt = "sehr gut";
else if (note == 2)
    beurt = "gut";
else if (note == 3)
    beurt = "befr.";
else if (note == 4)
    beurt = "genügend";
else if (note == 5)
    beurt = "nicht gen.";
else
    beurt = "?";

System.out.println(beurt);
```

```
int note = ...;
String beurt;

switch (note) {
    case 1: beurt = "sehr gut";
            break;
    case 2: beurt = "gut";
            break;
    case 3: beurt = "befr.";
            break;
    case 4: beurt = "genügend";
            break;
    case 5: beurt = "nicht gen.";
            break;
    default: beurt = "?";
}

System.out.println(beurt);
```

Aufgabe:
Wann ist welche Programmverzweigung besser?

Suchen Sie in Gruppen zu 2 oder 3 Personen eine Antwort auf diese Frage.