1.6.F Programmstrukturen/Abweisende Schleife – Ergänzungen und Bilder

1.6.F.1 Diagramme

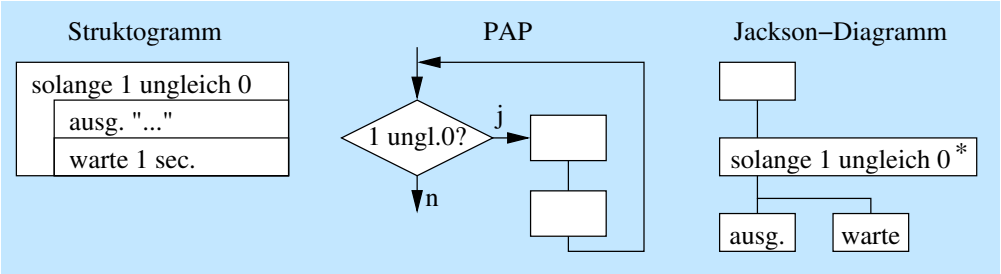

Abbildung 1: Kopfgesteuerte Schleife in Struktogramm, Flussdiagramm und Jackson-Diagramm

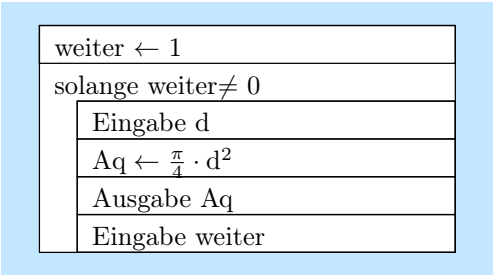

Abbildung 2: Struktogramm zu `while2.c`

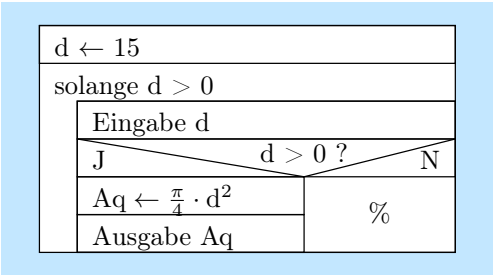

Abbildung 3: Struktogramm zu `while3.c`

1.6.F.2 Ärger mit dem Postfix-++-Operator

Der Operator ++ erhöht den Wert seines Operanden um eins. Setzt man den Operator im Quelltext *vor* einen Variablennamen, spricht man vom *Prefix-Operator* ++. Mit `++x` wird x um eins erhöht:

```
1  int x=3;
2  ++x;
3  // x ist danach 4
```

Wenn man `++x` an eine andere Variable zuweist, bekommt diese den neuen, erhöhten Wert von x:

```
1  int x=3;
```

```
2     int y;
3     y=++x; // x ist jetzt 4; 4 wird auch an y zugewiesen
```

Statt ++x kann man auch x++ schreiben. Dann spricht man vom *Postfix-Operator* ++. Auch hier wird mit ++x der Wert von x um eins erhöht:

```
1     int x=3;
2     x++;
3     // x ist danach 4
```

Aber warum gibt es zwei Möglichkeiten, x um eins zu erhöhen? Der Postfix-Operator hat eine Besonderheit: Bei Benutzung des Postfix-Operators mit x++ hat der Ausdruck x++ noch den *alten* Wert von x.

```
1     int x=3;
2     int y;
3     y=x++;
4     // x hat jetzt den neuen Wert 4
5     // aber an y wird 3 zugewiesen
```

Das kann so realisiert werden, dass der Wert zuerst ausgewertet wird und erst dann x erhöht wird.

Schwierig wird das dann, wenn man folgende Konstruktion baut:

```
1     int x=3;
2     x=x++;
3     // x hat jetzt den neuen Wert 4
4     // aber an x wird 3 zugewiesen
```

Was gibt das folgende Programm aus?

```
1  #include <stdio.h>
2  #include <time.h>
3  int main(void)
4  {
5      int punkte=0;
6      printf("Bitte Zahl eingeben: ");
7      scanf("%i", &punkte);
8
9      punkte = punkte++;
10     printf("Zahl wurde zugewiesen und um eins erhoeht.\n");
11
12     printf("Neuer Wert: %i\n", punkte);
13     return 0;
14 }
```

Terminal

```
schueler@debian964:~$ gcc -Wall postfix.c
postfix.c: In function 'main':
postfix.c:9:11: warning: operation on 'punkte' may be undefined
                         [-Wsequence-point]
    9 |     punkte = punkte++;
      |     ~~~~~~~^~~~~~~~~~
schueler@debian964:~$ a.out
Bitte Zahl eingeben: 100
Zahl wurde zugewiesen und um eins erhoeht.
Neuer Wert: 100
```

Wie die Fehlermeldung des Compilers es sagt: Das Verhalten in diesem Fall ist *undefiniert*[1]. Das liegt daran, dass dieselbe Variable in einem Ausdruck zweimal zugewiesen wurde und die Reihenfolge der Zuweisungen vom Standard nicht festgelegt wurde. In diesem Fall war es beabsichtigt, die Variable `punkte` um eins zu erhöhen, was aber fehlgeschlagen ist. Mit einem anderen Compiler wäre ein anderes Verhalten möglich.

Mit einfachen Maßnahmen kann man solche Fälle vermeiden; hier einige Tipps:

- Jede Zuweisung in eine eigene Zeile (meint hier: Anweisung) schreiben. Niemals versuchen, Zeilen zu sparen[2].
- Insbesondere `++x` und `x++` nur alleine in einer Anweisung verwenden. Dann ist es auch egal, ob man `++x` und `x++` schreibt.
- Auf den Postfix-Operator lieber verzichten: Er kostet nur Rechenzeit und behindert Werkzeuge, die den Quelltext analysieren wollen.

[1] Dieses Wort stammt aus der ANSI-Sprachdefinition von C und meint: Es kann alles passieren, vom Absturz des Programms über falsches Programmverhalten bis hin zum erwarteten Verhalten.

[2] Der Grund: Nur bei so genannten *Sequenzpunkten* weiß man, dass alle Zuweisungen abgeschlossen wurden. Der bekannteste Sequenzpunkt ist das Semikolon. Nach einem Semikolon kann man mit neuen Zuweisungen beginnen.