3.2 Formulierung algorithmischer Problemlösungen als Computerprogramm

Methode der strukturierten Programmierung

Nach dem Kennenlernen der Grundelemente von Algorithmen sollen nunmehr auch kompliziertere Algorithmen in übersichtlichen Programmen erstellt werden. Dies erfordert eine systematische und methodisch geschickte Aufbereitung des jeweiligen Problems. Eine sinnvolle Bearbeitung eines Problems kann in folgenden Schritten der so genannten strukturierten Programmierung geschehen (in Klammern stehen Erläuterungen und exemplarische Überlegungen zur Erstellung eines Programms zur Lösung quadratischer Gleichungen, das bereits weiter oben programmiert worden ist):

1. Analyse und Präzisierung der Problemstellung (In welcher Zahlenmenge soll $a \cdot x^2 + b \cdot x + c = 0$ lösbar sein? Soll eine spezielle Lösung für $a = 0$ angeboten werden?)
2. Festlegung der Objekte (Ein- und Ausgabedaten, Bildschirmgestaltung usw.)
3. Bereitstellung von Hilfsmitteln (Lösungsformel!)
4. Grobformulierung (Eingabe der Parameter, Anwendung der Lösungsformel, Ausgabe der Lösungen mit Kommentar)
5. Ausarbeitung der Details (vgl. Hauptproblem!; z. B. Vorgangsweise, wenn keine reelle Lösung existiert)
6. Graphische Darstellung (Struktogramm, Programmablaufplan)
7. Codieren nach festen Regeln (Erhalten der Struktur der Problemlösung, z. B. Gliederung in Sequenzen, Auswahlen und Wiederholungen)
8. Testen des Programms (Schreibtischtest oder Probeläufe)
9. Dokumentation des Programms (evt. Listing des Quelltextes; ferner Beschreibung von: Programmname, geeignete Hardware, nötige Zusatzgeräte, Programmablauf, theoretische Grundlagen, verwendete Variablen, nötige Kenntnisse, besondere Einzelheiten, Programmlaufzeit)

Ein Beispiel: Primzahlsuche

Die Prüfung einer Zahl auf Primzahleigenschaft ist ein ebenso bekanntes wie mühsames Problem, das sich in seinem Kern auf die Frage reduziert, ob eine gegebene natürliche Zahl weitere Teiler als 1 und sich selbst hat. Diese Prüfung ist gerade wegen seines gleichbleibenden Ablaufs für die Bearbeitung mit einem Computer besonders gut geeignet!

Nachfolgend einige Schritte zur Problemlösung:

1. Analyse und Präzisierung der Problemstellung: siehe oben!

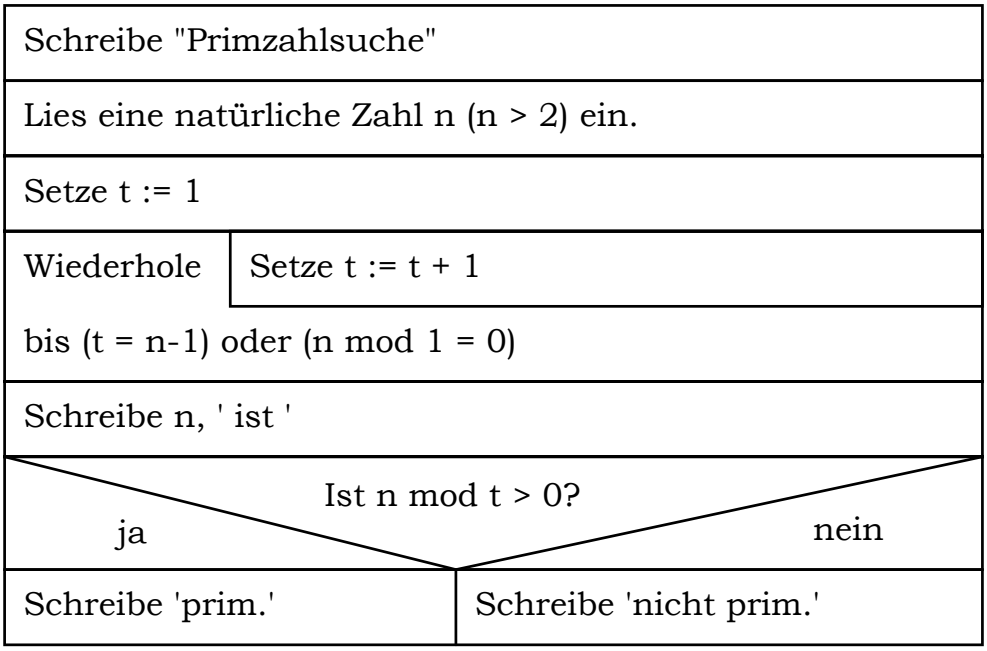
2. Festlegung der Objekte: Eingabe einer natürlichen Zahl; Ausgabe der Eigenschaft "prim" oder "nicht prim".

3. Bereitstellung von Hilfsmitteln: Man geht von der Überlegung aus, dass eine natürliche Zahl n > 2 dann keine Primzahl ist, wenn für mindestens eine Zahl $t \in \{2, 3, 4, ..., n-1\}$ der Quotient $\frac{n}{t}$ wieder eine natürliche Zahl ist. Dies ist äquivalent zur Aussage, dass n durch t ohne Rest teilbar ist, und wird in Pascal durch den Operator **MOD** geprüft.

Anmerkungen zur Codierung in Pascal:

1. Die Prüfung, ob t ein Teiler von n ist, wird mit dem Modulus-Operator **MOD** durchgeführt. Die Aussage $\frac{n}{t} \in N$ ist nämlich äquivalent zur Aussage n mod t = 0.
   Zugehörige Syntax:
   Integervariable1 **MOD** *Integervariable2*
   ermittelt den Divisionsrest.
2. Eine Prüfbedingung kann auch aus mehreren Teilbedingungen bestehen, die durch logische Verknüpfungen (**AND**, **OR**, usw.) miteinander verbunden sind. Die Teilbedingungen müssen in Klammern stehen.
   Syntax:
   IF *(Bedingung 1)* **AND/OR/XOR** *(Bedingung 2)* **THEN DO** *Anweisung*

6. Graphische Darstellung (Struktogramm):



Der Abbruch erfolgt also, wenn der Teiler t alle Zahlen von 2 bis n - 1 durchlaufen hat oder wenn der Teiler t die Zahl n teilt.

7. Codierung in Pascal:

```
program programm12_(primzahlpruefung_1);
uses crt;
var  n, t:     longint;
     antwort:  char;
begin
     repeat
          clrscr;
          writeln ('Primzahlprüfung 1');
          writeln;
          write ('Gib eine natürliche Zahl (n > 2) ein. ');
          readln (n);
          t := 1;
          repeat
               t := t + 1
          until (t = n-1) or (n mod t = 0);
          write (n, 'ist ');
          if n mod t > 0 then writeln ('prim.')
          else writeln ('nicht prim.');
          writeln;
          writeln ('Nochmals? (j/sonst. Taste)');
          antwort := readkey
     until antwort <> 'j'
end.
```

Diese Lösung ist wohl leicht verständlich, aber doch noch wesentlich verbesserungsfähig.