6.5 Einen Simulator programmieren

ZUSATZ

Die Registermaschine lässt sich in wenigen Zeilen nachbauen. Das Programm ist eine Liste von Befehlen, der Datenspeicher eine Liste von Zahlen, der Rest sind drei Variablen.

Python

```python
def simuliere(programm, eingaben):
    """programm: Liste von (Operation, Adresse). Gibt die Ausgaben zurueck."""
    daten = [0] * 100          # Datenspeicher
    akku = 0                   # Akkumulator
    zaehler = 0                # Befehlszaehler (Index im Programm)
    eingabe_nr = 0
    ausgaben = []

    while 0 <= zaehler < len(programm):
        operation, adresse = programm[zaehler]
        zaehler += 1                       # Schritt 5, ausser bei Spruengen

        if operation == "INP":
            daten[adresse] = eingaben[eingabe_nr]
            eingabe_nr += 1
        elif operation == "LDA":
            akku = daten[adresse]
        elif operation == "LDK":
            akku = adresse
        elif operation == "STA":
            daten[adresse] = akku
        elif operation == "ADD":
            akku += daten[adresse]
        elif operation == "SUB":
            akku -= daten[adresse]
        elif operation == "MUL":
            akku *= daten[adresse]
        elif operation == "DIV":
            akku //= daten[adresse]
        elif operation == "JMP":
            zaehler = adresse - 1
        elif operation == "JEZ" and akku == 0:
            zaehler = adresse - 1
        elif operation == "JNE" and akku != 0:
            zaehler = adresse - 1
        elif operation == "JLZ" and akku < 0:
            zaehler = adresse - 1
        elif operation == "JGZ" and akku > 0:
            zaehler = adresse - 1
        elif operation == "JLE" and akku <= 0:
            zaehler = adresse - 1
        elif operation == "JGE" and akku >= 0:
            zaehler = adresse - 1
        elif operation == "OUT":
            ausgaben.append(akku if adresse == 0 else daten[adresse])
        elif operation == "HLT":
            break

    return ausgaben
```

```
50
51
52 MINIMUM = [("INP", 1), ("INP", 2), ("LDA", 1), ("SUB", 2), ("JGE", 8),
53            ("LDA", 1), ("JMP", 9), ("LDA", 2), ("OUT", 0), ("HLT", 99)]
54
55 print(simuliere(MINIMUM, [11, 13]))      # [11]
56 print(simuliere(MINIMUM, [20, 7]))       # [7]
```

Beachten Sie die Zeile `zaehler = adresse - 1` bei den Sprüngen: Die Registermaschine zählt ihre Programmzeilen ab 1, eine Python-Liste ihre Plätze ab 0. Ein Sprung nach Zeile 8 muss deshalb auf den Listenindex 7 führen. Das vorangegangene `zaehler += 1` spielt dabei keine Rolle — die Zuweisung überschreibt es ja. Genau solche Verschiebungen um eins übersieht man beim ersten Versuch, und der Simulator landet dann in der falschen Zeile oder in einer Endlosschleife. Die Bedingung `0 <= zaehler` in der Schleife fängt wenigstens den Fall ab, dass ein fehlerhafter Sprung ins Negative führt — ohne sie würde Python stillschweigend von hinten zählen.

Object Pascal

```
1  program Registermaschine;
2
3  type
4    TBefehl = record
5      Operation: string[3];
6      Adresse: Integer;
7    end;
8
9  const
10   PROGRAMM: array[1..10] of TBefehl = (
11     (Operation: 'INP'; Adresse: 1), (Operation: 'INP'; Adresse: 2),
12     (Operation: 'LDA'; Adresse: 1), (Operation: 'SUB'; Adresse: 2),
13     (Operation: 'JGE'; Adresse: 8), (Operation: 'LDA'; Adresse: 1),
14     (Operation: 'JMP'; Adresse: 9), (Operation: 'LDA'; Adresse: 2),
15     (Operation: 'OUT'; Adresse: 0), (Operation: 'HLT'; Adresse: 99));
16
17 var
18   daten: array[0..99] of Integer;
19   akku, zaehler, i: Integer;
20   b: TBefehl;
21
22 begin
23   for i := 0 to 99 do daten[i] := 0;
24   akku := 0;
25   zaehler := 1;
26
27   while (zaehler >= 1) and (zaehler <= Length(PROGRAMM)) do
28   begin
29     b := PROGRAMM[zaehler];
30     Inc(zaehler);                                  { Schritt 5 }
31
32     if b.Operation = 'INP' then ReadLn(daten[b.Adresse])
33     else if b.Operation = 'LDA' then akku := daten[b.Adresse]
34     else if b.Operation = 'LDK' then akku := b.Adresse
35     else if b.Operation = 'STA' then daten[b.Adresse] := akku
36     else if b.Operation = 'ADD' then akku := akku + daten[b.Adresse]
37     else if b.Operation = 'SUB' then akku := akku - daten[b.Adresse]
38     else if b.Operation = 'MUL' then akku := akku * daten[b.Adresse]
```

```
39      else if b.Operation = 'DIV' then akku := akku div daten[b.Adresse]
40      else if b.Operation = 'JMP' then zaehler := b.Adresse
41      else if (b.Operation = 'JEZ') and (akku =  0) then zaehler := b.Adresse
42      else if (b.Operation = 'JNE') and (akku <> 0) then zaehler := b.Adresse
43      else if (b.Operation = 'JLZ') and (akku <  0) then zaehler := b.Adresse
44      else if (b.Operation = 'JGZ') and (akku >  0) then zaehler := b.Adresse
45      else if (b.Operation = 'JLE') and (akku <= 0) then zaehler := b.Adresse
46      else if (b.Operation = 'JGE') and (akku >= 0) then zaehler := b.Adresse
47      else if b.Operation = 'OUT' then
48        if b.Adresse = 0 then WriteLn(akku) else WriteLn(daten[b.Adresse])
49      else if b.Operation = 'HLT' then Break;
50    end;
51  end.
```

Die Pascal-Fassung zählt ab 1 wie im Skript, die Python-Fassung ab 0 wie die Liste — deshalb steht dort `adresse - 1` und hier nicht. Ein guter Anlass, beide Programme nebeneinander zu legen.

Arbeitsauftrag 4

16. Ergänzen Sie den Simulator um eine Ausgabe nach jedem Schritt: Befehlszähler, Befehl, Akkumulator. Damit haben Sie einen Einzelschrittmodus.
17. Bauen Sie eine Sicherung ein, die nach 10 000 Schritten abbricht, und erklären Sie, warum das nötig ist — und warum sie das Problem nicht löst, sondern nur umgeht. (Vergleichen Sie Abschnitt 4.4.)
18. Lassen Sie Ihre Programme aus 6.4 im Simulator laufen und vergleichen Sie mit Ihrer Handrechnung.
19. Erweitern Sie den Simulator um einen Befehlszähler für die Statistik: Wie viele Schritte braucht Ihr ggT-Programm für $a = 1071$, $b = 462$?
20. **Projekt:** Schreiben Sie einen kleinen Assembler, der ein Programm aus einer Textdatei im Format `LDA 01` einliest und in die interne Darstellung übersetzt.

Gymnasium Ottweiler