Matthias Jauernig
Kurs: 03In1

06.11.03

5. Aufgabenserie zu den Grundlagen der Informatik

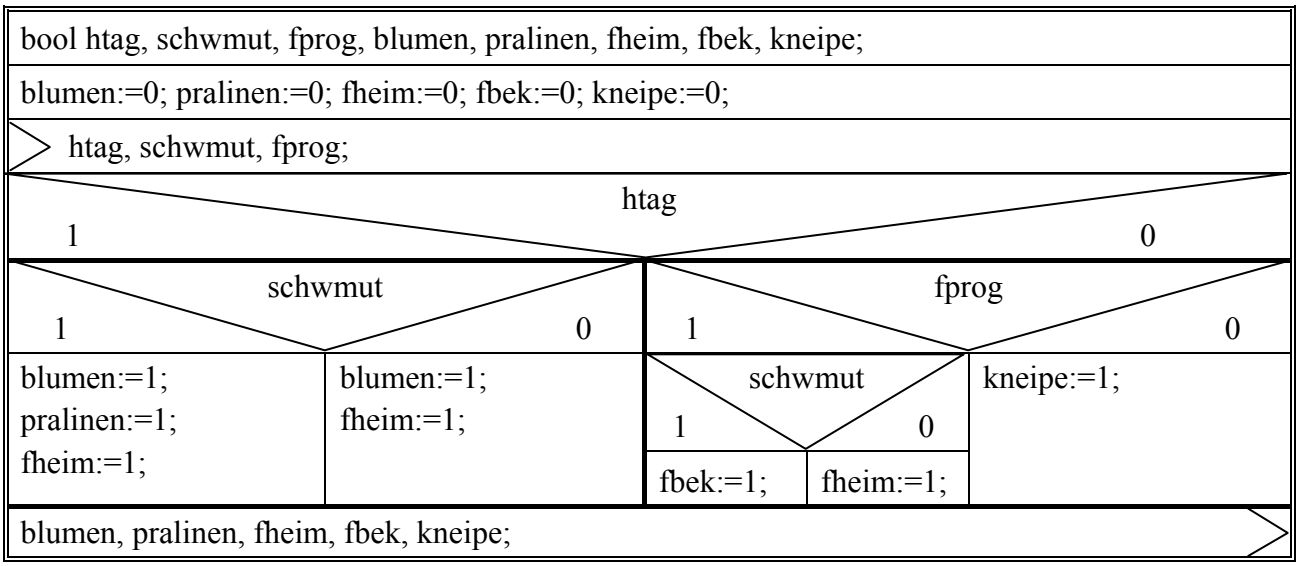
Abgabetermin: Mi, 12.11.03

Zu 13.) benutzte Variablen im Struktogramm und ihre Bedeutung:

htag	:=	Hochzeitstag
schwmut	:=	Schwiegermutter zu Besuch
fprog	:=	Fernsehprogramm interessant
blumen	:=	Blumen kaufen
pralinen	:=	Pralinen kaufen
fheim	:=	Fernsehen zu Hause
fbek	:=	Fernsehen bei Bekannten
kneipe	:=	Stammkneipe aufsuchen

⇒ diese Variablen haben jeweils den Wahrheitswert "wahr" oder "falsch", es sind damit boolesche Variablen (Datentyp: bool bzw. boolean)

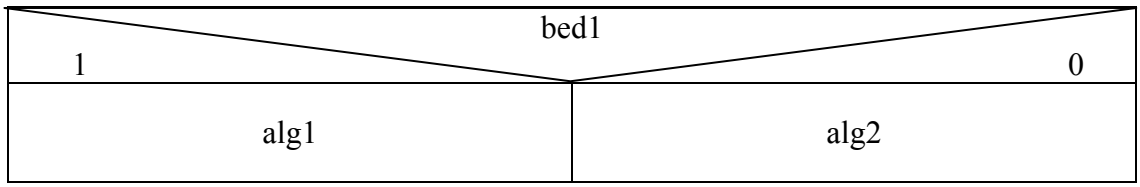
Anhand der gegebenen Entscheidungstabelle lässt sich folgendes
Struktogramm aufstellen:



Zu 14.) (1) Simulation der *zweiseitigen Auswahl* durch *Sequenz* und *anfangsgeprüfte Schleife:*

Es seien alg1, alg2 wohlstrukturierte Algorithmen, bed1 die zu überprüfende Bedingung und bed2 eine Art „Hilfsbedingung“, um die Simulation zu ermöglichen.

Mithilfe der zweiseitigen Auswahl lässt sich das Problem als Struktogrammelement folgendermaßen darstellen:



Mithilfe von Sequenz und anfangsgeprüfter Schleife lässt sich eine zweiseitige Auswahl struktogrammmäßig folgendermaßen erfassen:

<table>
<tr><td colspan="2">bool bed2:=1;</td></tr>
<tr><td colspan="2">while (bed2 && bed1)</td></tr>
<tr><td></td><td>alg1</td></tr>
<tr><td></td><td>bed2:=0;</td></tr>
<tr><td colspan="2">while bed2</td></tr>
<tr><td></td><td>alg2</td></tr>
<tr><td></td><td>bed2:=0;</td></tr>
</table>

⇒ Erklärung: mittels 2 anfangsgeprüfter Schleifen und Sequenzen wird hier die zweiseitige Auswahl simuliert, indem nur in die 1. Schleife hinein gegangen wird, wenn bed1 und bed2 wahr sind. Ist dies der Fall, so wird alg1 ausgeführt und die Hilfsvariable bed2:=0 gesetzt, um die Schleifenbedingung der 1. Schleife nach einmaligem Durchlauf falsch werden zu lassen und um zu verhindern, dass die 2. Schleife ausgeführt wird. Wurde die 1. Schleife nicht ausgeführt, so ist die Bedingung der 2. Schleife erfüllt: diese wird also durchlaufen, wobei alg2 ausgeführt und bed2:=0 gesetzt wird, um die Schleifenbedingung nach einmaligem Durchlaufen falsch werden zu lassen.

→ Anmerkung: In Programmiersprachen, wo dies möglich ist, könnte man die Schleifen natürlich auch mit „break“ nach einmaligem Durchlaufen abbrechen. Da hier aber durch die Aufgabenstellung nichts näher spezifiziert wurde, habe ich dies aufgrund größerer Verallgemeinerung weg gelassen.

(2) Simulation der *endgeprüften Schleife* durch *Sequenz* und *anfangsgeprüfte Schleife:*

Es sei alg ein wohlstrukturierter Algorithmus, bed1 die Schleifenbedingung, die falsch sein muss, damit die Schleife verlassen wird und bed2 wieder eine Art „Hilfsbedingung“, um die Simulation zu ermöglichen.

Mithilfe der endgeprüften Schleife lässt sich das Problem als Struktogrammelement folgendermaßen darstellen:

<table>
<tr><td></td><td>alg</td></tr>
<tr><td colspan="2">until bed1</td></tr>
</table>

Mithilfe von Sequenz und anfangsgeprüfter Schleife lässt sich eine endgeprüfte Schleife struktogrammmäßig folgendermaßen erfassen:

<table>
<tr><td colspan="2">bool bed2:=1;</td></tr>
<tr><td colspan="2">while (bed2 || !bed1)</td></tr>
<tr><td></td><td>alg</td></tr>
<tr><td></td><td>bed2:=0;</td></tr>
</table>

⇒ Erklärung: mittels einer anfangsgeprüften Schleife und Sequenzen wird hier die endgeprüfte Schleife simuliert, indem bed2 vor dem ersten Durchlauf der Schleife der Wert 1 zugewiesen wird, womit die Schleifenbedingung der anfangsgeprüften Schleife auf jeden Fall erfüllt ist, egal welchen Wahrheitswert bed2 besitzt. Der Schleifenkörper und somit auch „alg“ wird daher mindestens einmal ausgeführt, so wie es eben auch bei einer endgeprüften Schleife der Fall ist. Nun wird aber bed2:=0 gesetzt, womit die Schleifenbedingung nur noch erfüllt ist, wenn bed1 nicht erfüllt ist (*Achtung:* Negierung der Schleifenbedingung ist hier wegen des Austauschs von until [„bis“] gegen while [„solange“] notwendig!). Da es nach dem ersten Durchlauf in diesem Fall egal ist, ob die Bedingung am Anfang oder am Ende der Schleife überprüft wird, ist die Simulation der endgeprüften Schleife „erfolgreich“.

Zu 15.) Trace-Tabellen: die Beispielwerte für x und y können den Tabellen entnommen werden

x	y	z	x	y	z
2.321	13	1	1.1111	129	1
	12	2.321		128	1.1111
5.38704	6		1.234543	64	
29.0202	3		1.524097	32	
	2	67.3559	2.322871	16	
842.173	1		5.395732	8	
	0	***56725.3***	29.113923	4	
			847.620489	2	
			718460.493143	1	
				0	***798281.453931***

x	y	z	x	y	z
4	4	1	3.2	14	1
16	2		10.24	7	
256	1			6	10.24
	0	***256***	104.8576	3	
				2	1073.741824
			10995.116278	1	
				0	***11805916.2071***

⇒ aus den Tabellen folgt: nach Abarbeitung des Algorithmus besitzt z den Wert: $\mathbf{z=x^y}$

Beweis der Richtigkeit:

Für die äußere und auch innere while-Schleife lassen sich zwei Schleifeninvarianten (iv1) und (iv2) beschreiben, mit deren Hilfe man die Gültigkeit der Berechnung von x^y mit diesem Algorithmus beweisen kann.

1. Schleifeninvariante: (äußere Schleife)

Es seien j der Wert von x und k der Wert von y vor dem ersten Einstieg in die äußere while-Schleife (also die Anfangswerte von x und y).
Aus den Beispiel-Tracetabellen ist ersichtlich, dass die äußere Schleife immer dann beendet ist, wenn z ein neuer Wert zugewiesen wird.
Weiterhin ist erkennbar, dass $z*x^y$ gleich j^k ist immer dann, wenn der Schleifenkörper der äußeren Schleife erneut ausgeführt wird.

⇒ Schleifeninvariante **iv1:** $\mathbf{z * x^y = j^k}$

2. Schleifeninvariante: (innere Schleife)

Es seien m der Wert von x und n der Wert von y jeweils direkt nach dem Eintreten in die äußere Schleife (also noch vor dem Einstieg in die innere Schleife).
Mit diesen Werten m, n gilt nun:

⇒ Schleifeninvariante **iv2:** $\mathbf{x^y = m^n}$

Aufstellen von Behauptungen:

(B1) iv1 ist vor der Ausführung der äußeren Schleife richtig
(B2) iv2 ist jeweils vor Ausführung der inneren Schleife richtig
(B3) die äußere Schleife terminiert
(B4) die innere Schleife terminiert
(B5) gelte iv1 vor Ausführung des Schleifenkörpers der äußeren Schleife, so gelten sie auch danach
(B6) gelte y>0 und iv2 vor Ausführung des Schleifenkörpers der inneren Schleife, so gelten sie auch danach

Beweis der Behauptungen (B1)-(B6):

- zu (B1): vor Ausführung gilt: z=1, j=x, k=y und somit $z*x^y=1*j^k=j^k$ ⇒ (B1) wahr
- zu (B2): vor Ausführung gilt: m=x, n=y und somit $x^y=m^n$ ⇒ (B2) wahr
- zu (B3): y wird in der Schleife mindestens um 1 verringert, wodurch die Bedingung y>0 nach spätestens y Schleifendurchläufen nicht mehr erfüllt ist ⇒ (B3) wahr
- zu (B4): y wird in der inneren Schleife durch 2 geteilt; selbst wenn y eine Zweierpotenz darstellt, so ist die Schleifenbedingung „y:=gerade“ spätestens für y=1 nicht mehr erfüllt (2/2=1) ⇒ (B4) wahr
- zu (B5): Fallunterscheidung:
  Fall 1: y ist gerade: in diesem Fall wird die innere Schleife ausgeführt, wobei x^y erhalten bleibt, wie der nachfolgende Beweis von (B6) zeigt; weiterhin verhält es sich wie bei „y ist ungerade“, wie Fall 2 zeigt
  Fall 2: y ist ungerade: in diesem Fall gilt nach Ausführung der äußeren Schleife: y=y-1, z=z*x; eingesetzt in iv1: $(z*x)*x^{y-1} = z*x*x^{y-1} = z*x^y = j^k$
  ⇒ (B5) wahr
- zu (B6): nach Durchlauf der inneren Schleife gilt: x=x*x, y=y/2, eingesetzt in iv2: $(x*x)^{y/2} = (x^2)^{y/2} = x^y = m^n$; y>0, da innere Schleife terminiert, wenn y ungerade ⇒ (B6) wahr

⇒ **Daraus folgt: (B1) bis (B6) sind wahr. Das bedeutet, dass der vorgegebene Algorithmus bewiesenermaßen als Ergebnis $\underline{z=x^y}$ liefert.** **(q.e.d.)**

Was zeichnet diesen Algorithmus aus?

Der Algorithmus liefert, wie soeben bewiesen, als Ergebnis den Wert x^y. Trivial berechnen würde man diesen durch y-fache Multiplikation von x mit sich selbst.

Der vorliegende Algorithmus löst das Problem allerdings wesentlich eleganter und vor allem mit weniger Rechenaufwand, indem immer dann, wenn y eine gerade Zahl ist, der Term x^y zu $(x*x)^{y/2}$ umgeformt wird, wodurch auf Kosten nur einer einzigen Multiplikation der Exponent um die Hälfte geschrumpft ist.
Wie oft diese Umformung stattfindet, hängt vom Anfangswert von y ab, wobei es qualitativ auf den Algorithmus bezogen „schlechte“ und „gute“ Werte für y gibt.

Im Allgemeinen lässt sich jedoch sagen, dass mit diesem Algorithmus selbst große Potenzen zweier Zahlen vervielfacht schneller berechnet werden können als mit der Triviallösung.

C-Programm:

```
/*      Matthias Jauernig, 03In1; 05.11.03                          */
/*      Implementierung eines Algorithmus zur Berechnung von x^y,   */
/*      wenn x eine reelle und y eine natürliche Zahl               */

#include <stdio.h>

int main(void){
        long int y;
        double x, z=1.0;

        printf("\nBestimmung von x^y\n==================\n\n");

        printf("Reelle Zahl x eingeben: "); scanf("%lf", &x);
        do{
            printf("Natürliche Zahl y eingeben:  ");
            scanf("%ld", &y);
        }while(y<=0 && printf("Zahl muss groesser 0 sein!\n\n"));

        while(y>0){
            while(y%2==0){
                  y/=2;
                  x*=x;
            }
            y--;
            z*=x;
        }

        printf("z=%lf\n", z);

        return 0;
}
```