Programmieren mit dem TI-84+

Theorie

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Das erste Programm 1

1.2 Programme bearbeiten 2

1.3 Fehlerarten 3

1.4 Aufgaben 4

2 Sequenz **5**

2.1 Ein Umrechnungsprogramm 5

2.2 Aufgaben 6

3 Alternative **8**

3.1 Wahrheitswerte 8

3.2 Bedingte Anweisungen 9

3.3 Verzweigungen 11

3.4 Aufgaben 11

4 Iteration **14**

4.1 Die For-Schleife 14

4.2 Die While-Schleife 14

4.3 Die Repeat-Schleife 15

4.4 Aufgaben 15

5 Abstraktion **18**

5.1 Aufgaben 19

6 Listen **21**

6.1 Aufgaben 22

Kurzreferenz **24**

1 Einleitung

1.1 Das erste Programm

Ein neues Programm erzeugt man mit

[prgm] NEW 1:Create New [enter]

Der Programmname besteht aus maximal 8 Zeichen und darf weder mit einer Ziffer beginnen noch Sonderzeichen enthalten.

Beim Eintippen des Programmnamens befindet sich der Rechner im `alpha`-Modus, weshalb die Buchstaben direkt mit den entsprechenden Tasten eingetippt werden können.

Unser erstes Beispielprogramm soll `HELLO` heissen. Die Eingabe des Programmnames muss mit [enter] bestätigt werden.

Danach befinden wir uns im *Programmeditor*.

```
PROGRAM:HELLO
:
```

Das Programm soll mit dem Befehl `Disp` (von *display*) eine Begrüssung am Bildschirm ausgeben.

[prgm] I/O 3:Disp [enter]

Damit der Rechner den folgenden Text nicht als Variable(n) deutet, muss er in Anführungs- und Schlusszeichen eingeschlossen werden. Jetzt muss vorher die [alpha]-Taste gedrückt bzw. der `A-lock`-Modus eingeschaltet werden.

```
PROGRAM:HELLO
:Disp "HELLO WORLD"
:
```

Das Leerzeichen befindet sich über der [0]-Taste und die Gänsefüsschen über der [+]-Taste. Leere Programmzeilen spielen keine Rolle. Jede Eingabe wird sofort gespeichert, weshalb kein spezieller „Speichern“-Befehl nötig ist.

Nachdem wir unser erstes Programm geschrieben haben, müssen wir den Programm-Editor vor der Programmausführung mit

[2nd] quit

verlassen. Wie bereits erwähnt, wird das Programm automatisch im aktuellen Zustand gespeichert.

Um das Programm auszuführen, drücken wir

[prgm] EXEC [▼] (...) HELLO [enter] [enter]

wobei wir in der Liste mit den Programmnamen mit dem Cursor auf den Namen `HELLO` fahren und mit [enter] auswählen.

Mit dem ersten [enter] wird das Programm am Haupbildschirm angezeigt, mit dem zweiten [enter] wird es ausgeführt und die folgende Ausgabe erscheint auf dem Bildschirm.

```
HELLO WORLD
                    Done
```

1.2 Programme bearbeiten

Die `clear`-Taste

- Mit der [clear]-Taste löscht man den Inhalt der gesamten Zeile, auf der sich der Cursor befindet.

  Leider lässt sich diese Aktion nicht rückgängig machen.

- Wenn man innerhalb eines Programms ungewollt ein Menü öffnet (`prgm`), so kann man es mit der [clear]-Taste wieder schliessen, ohne dass dabei Programmcode verloren geht.

Die `del`-Taste

Mit der [del]-Taste löscht man das Zeichen, auf dem der Cursor steht. Ebenso kann man damit überflüssige Zeilenschaltungen entfernen, was man mit `clear` nicht schafft.

Die `ins`-Funktion

Hat man versehentlich Programmcode gelöscht oder sollte man weiteren Code in ein Programm einfügen, so geht dies mit

[2nd] [ins]

Man beachte, dass sich mit der [enter] zusätzliche (Leer-)Zeilen einfügen lassen.

Ein Programm löschen

Ein bestehendes Programm ABC löscht man mit

[2nd] [mem] `2:Mem Mgmt/Del... 7:Prgm... ABC` [del]

und dann entweder mit `1:No` abbrechen oder mit `2:Yes` bestätigen.

Ein Programm „kopieren“

Will man das Programm XYZ in das Programm ABC einfügen, so setzt man im Programm ABC den Cursor an die Stelle, an der man das Programm `XYZ` haben möchte und gibt die folgenden Tastenbefehle ein:

[2nd] [rcl] [prgm] `EXEC XYZ` [enter] [enter]

Dies ist übrigens auch die einzige Möglichkeit, ein Programm „umzubenennen“.

1.3 Fehlerarten

Syntaxfehler

Eine Computerprogramm muss syntaktisch korrekt sein, bevor es vom Prozessor ausgeführt wird, da es sonst den Rechner in einen nicht definierten Zustand versetzen kann.

Daher überprüft der TI-84 Plus vor der Ausführung jeder Programmzeile, ob sie seinen Syntaxregeln genügt. Das bedeutet zum Beispiel, dass dort, wo ein Komma stehen muss, auch ein Komma steht oder dass zu jedem öffnenden Gänsefüsschen auch ein schliessendes vorhanden ist. Ebenfalls verboten sind zusätzliche Leerzeichen ausserhalb von Zeichenketten.

Der Rechner meldet solche Fehler mit

```
ERR: SYNTAX
1:Quit
2:Goto
```

In der Regel sollte man die Option `2:Goto` wählen, da dann das Programm automatisch geöffnet und der Cursor in die Zeile gesetzt, wo der Rechner den Fehler vermutet.

Laufzeitfehler

Auch wenn ein Programm syntaktisch korrekt geschrieben wurde, gibt es noch weitere Fehlerquellen. Eine davon besteht darin, dass eine Rechnung oder eine Benutzereingabe *zur Laufzeit* des Programms einen Fehler verursacht.

Diese Laufzeitfehler entstehen beispielsweise dann, wenn der Wert einer Variablen eine Division durch Null oder einen negativen Radikanden verursacht.

Der Programmierer sollte selber dafür sorgen, dass falsche Eingaben abgefangen werden, bevor sie zu einem Programmabbruch führen.

Wie man das macht, werden wir später sehen.

Logische Fehler

Schliesslich kann es sein, dass der Programmierer ein Programm geschrieben hat, das syntaktisch korrekt ist und keine Laufzeitfehler verursacht aber dennoch nicht richtig funktioniert.

Dies liegt dann daran, dass er er ein falsches Verfahren zur Lösung der Aufgabe verwendet hat oder sich bei einer Formel vertippt hat.

Um solche Fehler zu erkennen, sollte man die zu lösende Programmieraufgabe für einige Beispiele von Hand durchrechnen oder bekannte Lösungen heranziehen, um logische Fehler im Programm zu erkennen.

1.4 Aufgaben

Aufgabe 1.1

Wie lauten die Regeln für gültige Programmnamen?

- maximal 8 Zeichen
- Buchstaben und Ziffern
- keine Ziffer an erster Stelle

Aufgabe 1.2

Warum müssen in Programmen Zeichenketten wie `"HELLO"` in Anführungs- und Schlusszeichen eingeschlossen werden?

Sonst würde sie der TI-84 als (Produkt von) Variablen auffassen.

Aufgabe 1.3

Welche Art von Programmfehler verursacht das folgende Programm, wenn der Benutzer die Werte $A = 5$ und $B = 0$ eingibt. Verwende den richtigen Fachausdruck.

```
PROGRAM:RECHNUNG
:Prompt A
:Prompt B
:A/B→C
:Disp C
```

einen *Laufzeitfehler* (Division durch Null)

2 Sequenz

Ohne Steuerbefehle wird ein Programm sequentiell, das heisst zeilenweise von oben nach unten abgearbeitet.

2.1 Ein Umrechnungsprogramm

Unser zweites Programm soll eine Temperaturangabe von Grad Celsius in Grad Fahrenheit umrechnen:

prgm `NEW` 1:Create New enter C2F enter

Dann fordern wir den Benutzer auf, eine Eingaben in Grad Celsius zu machen:

```
PROGRAM:C2F
:Input "°CELSIUS: ",C
```

- `Input`: prgm `I/O` `1:Input` enter
- Grad-Symbol: 2nd `[angle]` `1:`°enter

Der Variablename `C` wurde frei gewählt und enthält später die vom Benutzer eingegebene Zahl.

Die Umrechnung von Grad Celsius (C) in Grad Fahrenheit (F) erfolgt nach der Formel

$$F = \frac{9}{5} \cdot C + 32$$

Diese Rechnung lässt sich direkt in das Programm einbauen:

```
PROGRAM:C2F
:Input "°CELSIUS: ",C
:9/5*C+32→F
```

Schliesslich müssen wir das Ergebnis noch ausgeben:

```
PROGRAM:C2F
:Input "°CELSIUS: ",C
:9/5*C+32→F
:Disp "°FAHRENHEIT: ",F
:
```

wobei `Disp` nach Betätigung der prgm-Taste wieder aus derm `I/O`-Menü geholt werden kann.

Das Programm `C2F` testen wir mit den folgenden Werten:

°Celsius	°Fahrenheit
0	32
100	212
−273.15	−459.67

2.2 Aufgaben

Aufgabe 2.1

Welche Ausgabe macht das das folgende Programm?

```
:5→A
:3→B
:A+B→C
:B+C→A
:Disp A
```

11

Aufgabe 2.2

Schreibe das Programm `NOTE` zur Berechnung von Prüfungsnoten auf der Basis einer linearen Notenskala:

$$N = \frac{P}{M} \cdot 5 + 1$$

wobei P: erreichte Punktzahl
M: Maximalpunktzahl (für Note 6)
N: Note

Testdaten:

	Test 1	Test 2	Test 3	Test 4
P	15	22	0	21
M	25	25	20	20
N	4	5.4	1	6.25

```
PROGRAM:NOTE
:Input "PUNKTE: ",P
:Input "MAXIMUM (6): ",M
:P/M*5+1→N
:Disp "NOTE:",N
```

Aufgabe 2.3

Schlage in der Formsammlung nach, wie man die Mantellinie (m), die Oberfläche (S) und das Volumen (V) eines geraden Kreiskegels aus dem Grundkreisradius (r) und der Höhe (h) berechnet.

Schreibe ein Programm `KEGEL` das diese drei Werte aus Radius und Höhe berechnet und ausgibt.

Testdaten (auf 2 Nachkommastellen gerundet):

	Kegel 1	Kegel 2	Kegel 3	Kegel 4
r	2	0	3	4
h	0	7	4	3
m	2	7	5	5
S	25.13	0	75.40	113.10
V	0	0	37.70	50.27

```
PROGRAM:KEGEL
:Input "RADIUS: ",R
:Input "HOEHE: ",H
:√(R²+H²)→M
:π*R*(R+M)→S
:R²*π*H/3→V
:Disp "MANTELLINIE:",M
:Disp "OBERFLAECHE:",S
:Disp "VOLUMEN:",V
```

Aufgabe 2.4

Schreibe ein Programm `HERON` mit dem man den Flächeninhalt F eines Dreiecks aus seinen Seitenlängen a, b und c in zwei Schritten bestimmt:

- $s = \dfrac{a+b+c}{2}$ (halber Umfang)
- $F = \sqrt{s(s-a)(s-b)(s-c)}$

Testdaten (auf 2 Nachkommastellen gerundet):

	Dreieck 1	Dreieck 2	Dreieck 3	Dreieck 4
a	3	1	5	2
b	4	1	2	6
c	5	1	7	3
F	6	0.43	0	Fehler (warum?)

```
PROGRAM:HERON
:Input "SEITE A: ",A
:Input "SEITE B: ",B
:Input "SEITE C: ",C
:(A+B+C)/2→S
:√(S(S-A)(S-B)(S-C))→F
:Disp "FLAECHE:",F
```

3 Alternative

Bisher wurden unsere Programme vom Beginn bis zum Ende zeilenweise verarbeitet. Wir werden jetzt Anweisungen kennen lernen, mit denen der Programmablauf vom Eintreten bestimmter Zustände abhängig gemacht werden kann.

3.1 Wahrheitswerte

Mathematische Aussagen sind entweder *wahr* oder *falsch*.

Solche Wahrheitswerte entstehen beim Vergleich von Zahlen mit *Vergleichsoperatoren*.

- $3 = 5$ falsch
- $3 \neq 5$ wahr
- $8 > 5$ wahr
- $4 \leq 4$ wahr
- $7 < 7$ falsch
- $3 \geq 5$ falsch

Die Vergleichsoperatoren findet man im Menü 2nd [test] `TEST`.

Anstelle von *wahr*/*falsch* gibt der TI-84 die Werte `1`/`0` aus.

Umgekehrt wird `0` als *falsch* und *jede Zahl* $\neq$ `0` als *wahr* interpretiert.

Verknüpfung von Wahrheitswerten

Mit *logischen Operatoren* lassen sich aus bestehenden Aussagen neuen Aussagen bilden:

Operator	Symbol	TI-84 Plus
Negation	$\neg$	`not(...)`
Konjunktion (UND)	$\wedge$	`and`
Disjunktion (ODER)	$\vee$	`or`
Kontravalenz (exklusives ODER)	$\dot{\vee}$	`xor`

Die logischen Operatoren findet man im Menü 2nd [test] `LOGIC`.

Die logische Verneinung

A	not(A)
1	0
0	1

Das logische ODER

Das logische ODER ist genau dann falsch, wenn beide Operanden falsch sind.

A	B	A or B
1	1	1
1	0	1
0	1	1
0	0	0

Das logische UND

Das logische UND ist genau dann wahr, wenn beide Operanden wahr sind.

A	B	A and B
1	1	1
1	0	0
0	1	0
0	0	0

Das logische exklusive ODER

Das logische exklusive (ausschliessende) ODER ist genau dann wahr (falsch), wenn genau einer der Operanden wahr (falsch) ist.

A	B	A xor B
1	1	0
1	0	1
0	1	1
0	0	0

3.2 Bedingte Anweisungen

If-Then

Hat die nach `If` stehende Bedingung den Wahrheitswert *true* ($\neq 0$), dann wird der darauffolgende, von `Then` und `End` eingeschlossene Anweisungsblock ausgeführt. Ist der Wahrheitswert der Bedingung *false* ($= 0$), so wird dieser Block übersprungen und das Programm nach dem Schlüsselwort `End` fortgesetzt.

```
:...
:If <Bedingung>
:Then
:<Anweisung(en) falls Bedingung wahr>
:End
:...
```

`If-Then`-Anweisungen können geschachtelt werden.

Beispiel

Welche Ausgabe macht das folgende Programmfragment?

```
:12→X
:If (X>10)
:Then
:X+2→X
:End
:Disp X
```

Ausgabe: 14

Beispiel

Welche Ausgabe macht das folgende Programmfragment?

```
:12→A
:If (A<10)
:Then
:A-4→A
:End
:Disp A
```

Ausgabe: 12

If

Sofern nach dem If nur eine einzelne Anweisung folgt, kann auch eine kürzere Syntax verwendet werden:

```
:...
:If <Bedingung>
:<genau eine Anweisung falls Bedingung wahr>
:<Fortsetzung des Programms>
:...
```

Diese Variante ist aber mit Vorsicht zu verwenden, da sie gerne zu Fehlern führt.

Beispiel

Welche Ausgabe macht das folgende Programmfragment?

```
:5→A          A=5
:3→B          A=5, B=3
:If A<10      wahr
:B+2→B        B=5
:If B<5       falsch
:A-4→A        überspringen
:If A*B>20    A*B=25>20 wahr
:B+1→B        B=6
:Disp A+B     A+B=5+6=11
```

Ausgabe: 11

3.3 Verzweigungen

If-Then-Else

Folgt `Else` nach `If-Then`, so wird eine Gruppe von Befehlen ausgeführt, wenn die Bedingung falsch (=0) ist. End markiert das Ende der Befehlsgruppe.

```
:...
:If <Bedingung>
:Then
:<Anweisung(en) falls Bedingung wahr>
:Else
:<Anweisung(en) falls Bedingung falsch>
:End
:...
```

`If-Then-Else`-Anweisungen können geschachtelt werden.

Beispiel

Welche Ausgabe macht das folgende Programmfragment?

```
:5→A
:3→B
:If (A+B>10)
:Then
:A+B→A
:Else
:A-B→A
:End
:Disp A
```

Ausgabe: 2

3.4 Aufgaben

Aufgabe 3.1

(a) `1 and 0` ⇒ 0

(b) `0 xor 1` ⇒ 1

(c) `not(0)` ⇒ 1

(d) `0 or 0` ⇒ 0

(e) `1 and 1` ⇒ 1

(f) `1 or 0` ⇒ 1

(g) `not(0 and 1)` ⇒ 1

(h) `1 xor not(0)` ⇒ 0

Aufgabe 3.2

(a) `(5 > 7) or (3 < 8)` ⇒ 1

(b) `not(7 ≤ 7)` ⇒ 0

(c) `not((0 or 0) and (1 and 0))` ⇒ 1

(d) `(1 xor 0) xor (0 xor 1)` ⇒ 0

(e) `not(5*7)` ⇒ 0

(f) `(0 or (0 or (0 or 1)))` ⇒ 1

(g) `(0 and (0 and (0 and 1)))` ⇒ 0

(h) `(2*3-6) or (8 ≠ 9)` ⇒ 1

Aufgabe 3.3

Schreibe ein Programm `DREIECK`, das den Benutzer auffordert, drei Seitenlängen `A`, `B` und `C` einzugeben und dann eine Meldung ausgibt, ob ein solches Dreieck existiert oder nicht.

Hinweise:

- Anstelle von

  `Disp "A? ",A`

  kann der gleichwertige Befehl

  `Prompt A`

  aus dem `I/O`-Menü verwendet werden.

- Ein Dreieck ist sinnvoll definiert, wenn es die *Dreiecksungleichung* erfüllt; d.h. wenn die Summe zweier Seitenlängen immer grösser als die dritte Seitenlänge ist.

Testdaten:

	Dreieck 1	Dreieck 2	Dreieck 3	Dreieck 4
a	3	1	5	2
b	4	1	2	6
c	5	1	7	3
definiert	ja	ja	nein	nein

```
:PROGRAM:DREIECK
:Prompt A,B,C
:If A+B>C and B+C>A and C+A>B
:Then
:Disp "DREIECK"
:Else
:Disp "KEIN DREIECK"
:End
```

Aufgabe 3.4

Schreibe ein Programm `QUADGLG`, das den Benutzer auffordert, die drei Koeffizienten a, b und c der quadratischen Gleichung

$$ax^2 + bx + c = 0$$

einzugeben, daraus die Diskriminante $D = b^2 - 4ac$ bestimmt und falls $D \geq 0$ gilt, die Lösungen

$$x_1 = \frac{-b + \sqrt{D}}{2a} \qquad x_2 = \frac{-b - \sqrt{D}}{2a}$$

berechnet und ausgibt. Andernfalls ist die Meldung auszugeben, dass die Gleichung keine reellen Lösungen hat.

```
:PROGRAM:QUADGLG
:Disp "AX²+BX+C=0"
:Prompt A,B,C
:B²-4AC→D
:If (D≥0)
:Then
:Disp "X1=",(-B+√(D))/2/A
:Disp "X2=",(-B-√(D))/2/A
:Else
:Disp "KEINE REELLEN LOESUNGEN"
:End
```

Aufgabe 3.5

Schreibe ein Programm mit dem Namen `LEAPYEAR`, das nach einer Jahreszahl fragt und bestimmt, ob es sich bei diesem Jahr um ein Schaltjahr handelt oder nicht.

Schaltjahre erfüllen die folgende Bedinung:

Die Jahreszahl ist durch 4 aber nicht durch 100 teilbar oder die Jahreszahl ist durch 400 teilbar.

```
:PROGRAM:LEAPYEAR
:Input "JAHR=",J
:If (remainder(J,4)=0
 and remainder(J,100)≠0)
 or remainder(J,400)=0
:Then
:Disp "SCHALTJAHR"
:Else
:Disp "KEIN SCHALTJAHR"
:End
```

4 Iteration

In diesem Kapitel geht es darum, wie man Programmteile wiederholt ausführen kann.

4.1 Die For-Schleife

`For` führt Schleifen aus und setzt eine Variable von einem Anfangswert schrittweise zu einem Endwert nach oben. Die Schrittweite ist optional (Voreinstellung ist 1) und kann auch negativ sein (Ende<Anfang). Ende ist der grösste oder kleinste Wert, der nicht überschritten werden darf. End markiert das Ende der Schleife. For-Schleifen können geschachtelt werden.

```
:For(<Variable>,<Anfang>,<Ende>[,<Schrittweite>])
:<Anweisung(en)>
:End
```

Beispiel

Das folgende Programmfragment berechnet die Summe der ersten 100 natürlichen Zahlen mit einer For-Schleife.

```
:0→S
:For(K,1,100)
:S+K→S
:End
:Disp "SUMME:",S
```

4.2 Die While-Schleife

`While` führt den Schleifenblock aus, so lange die Bedingung wahr ist. Die Bedingung wird beim Durchlaufen von While getestet. Ist die Bedingung wahr ($\neq 0$), führt das Programm eine Gruppe von Befehlen aus. End markiert das Ende der Gruppe. Ist die Bedingung falsch ($= 0$), führt das Programm die Befehle aus, die nach End stehen. While-Anweisungen können geschachtelt werden.

```
:While <Bedingung>
:<Anweisung(en), wenn Bedingung wahr>
:End
```

Beispiel

Das folgende Programmfragment berechnet die Summe der ersten 100 natürlichen Zahlen mit einer While-Schleife.

```
:0→S:0→K
:While(K<101)
:S+K→S:K+1→K
:End
:Disp "SUMME: ",S
```

4.3 Die Repeat-Schleife

Repeat wiederholt den Schleifenblock, *bis* die Bedingung wahr ($\neq 0$) ist. `Repeat` ist mit `While` vergleichbar, aber die Bedingung wird erst nach dem Durchlaufen von End überprüft. Daher wird die Befehlsgruppe immer mindestens einmal ausgeführt. `Repeat`-Anweisungen können verschachtelt werden.

```
:Repeat <Bedingung>
:<Anweisung(en), bis Bedingung wahr>
:End
```

Beispiel

Das folgende Programmfragment bittet den Benutzer so lange Zahlen einzugeben, bis er die Zahl Null eingibt. Dann wird die Schleife beendet und die Summe ausgegeben.

```
:0→S
:Repeat X=0
:Prompt X
:S+X→S
:End
:Disp "SUMME:",S
```

4.4 Aufgaben

Aufgabe 4.1

Schreibe ein Programm `FAKULT`, das die Fakultät $n!$ einer natürlichen Zahl $n \in \mathbb{N}$ mit einer For-Schleife berechnet.

```
PROGRAM:FAKULT
:Prompt N
:1→F
:For(I,1,N)
:F*I→F
:End
:Disp F
```

Aufgabe 4.2

Schreibe ein Programm `ISPRIME`, das eine natürliche Zahl $n > 1$ als Parameter entgegennimmt und feststellt, ob die Zahl prim ist, d. h. nur durch 1 und durch sich selbst ohne Rest teilbar ist.

Hinweise:

- Verwende die Funktion `remainder(N,K)`, die den Rest der Division von `N` durch `K` bestimmt.
- Es genügt, die Teilbarkeit von 2 bis und mit $\sqrt{N}$ zu prüfen.
- Mit `Stop` aus dem CTRL-Menü lässt sich ein Programm sofort anhalten.

```
PROGRAM:ISPRIME
:Prompt N
:2→K
:While K≤√(N)
:If remainder(N,K)=0
:Then
:Disp "NOT PRIME"
:Stop
:End
:K+1→K
:End
:Disp "PRIME"
```

Aufgabe 4.3

Schreibe ein Programm `GUESS`, das mit `randInt(1,100)` eine Pseudozufallszahl zwischen 1 und 100 erzeugt und den Benutzer so lange nach einer Zahl fragt, bis er sie erraten hat. Dabei informiert das Programm nach jedem Fehlversuch, ob die geratene Zahl zu klein oder zu gross ist.

Darüber hinaus ist eine Variable zu definieren, welche die Anzahl der Versuche zählt und diese nach dem Erraten der Zahl ausgibt.

```
PROGRAM:GUESS
:randInt(1,100)→C
:0→T
:Repeat N=C
:Prompt N
:If N<C
:Disp "ZU KLEIN"
:If N>C
:Disp "ZU GROSS"
:T+1→T
:End
:Disp "GESCHAFFT!"
:Disp "VERSUCHE:",T
```

Aufgabe 4.4

Ein eher seltsame Aufgabe ist die folgende: Schreibe ein Programm mit dem Namen `COLLATZ`, das den Benutzer auffordert, eine natürliche Zahl `N` einzugeben. Daraus wird wie folgt eine neue Zahl berechnet:

- Falls `N` gerade ist, ersetze `N` durch `N/2`.
- Falls `N` ungerade ist, ersetze `N` durch `3N+1`.

Dieser Vorgang wird so lange wiederholt, bis `N=1` ist. Dabei wird nach jedem Durchlauf die Zahl `N` ausgegeben.

Das besondere an dieser Folge, die übrigens Collatz-Folge genannt wird, ist, dass man bisher noch keinen Startwert finden konnte, von dem aus dem man das Ziel `N=1` nicht erreicht. Es ist jedoch noch niemandem gelungen, dies zu beweisen.

```
PROGRAM:COLLATZ
:Prompt N
:While N≠1
:If remainder(N,2)=0
:Then
:N/2→N
:Else
:3N+1→N
:End
:Disp N
:End
```

5 Abstraktion

Unterprogramme (Subroutinen, Prozeduren, Funktionen) sind wichtige Bausteine, um komplexere Programme zu schreiben. Sie erlauben es, das Gesamtprogramm aus mehreren Teilprogrammen aufzubauen, so dass der Programmierer den Überblick behält.

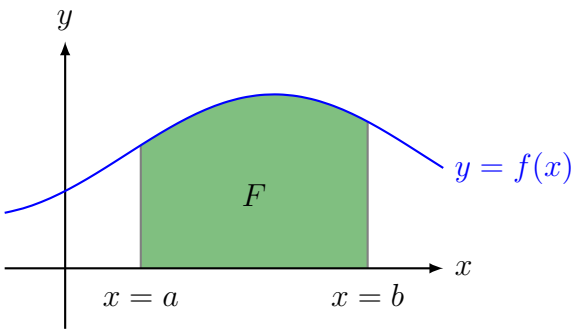
Bei einem sorgfältigen Design der Unterprogramme können diese leicht durch verbesserte Versionen ausgetauscht werden, ohne dass das gesamte Programm neu geschrieben werden muss.

prgm

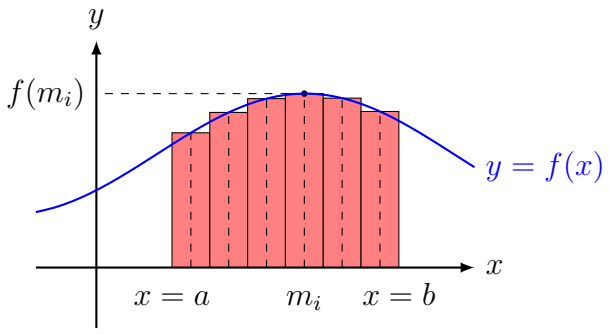
Mit `prgm` werden andere Programme als Unterprogramme ausgeführt. Bei Auswahl von `prgm` wird der Befehl an der Cursorposition eingefügt. Unmittelbar danach gibt man den Programmnamen ein. Die Verwendung von `prgm` entspricht der Auswahl bestehender Programme aus dem `PRGM EXEC`-Menü. Man kann übrigens auch den Namen eines Programms eingeben, das noch nicht erstellt wurde.

Beispiel: Das bestimmte Integral

Gesucht: Inhalt F der Fläche zwischen dem Graphen einer Funktion $y = f(x)$ mit nichtnegativen Werten, der x-Achse sowie den Grenzen $x = a$ und $x = b$.



Ersetze die Fläche durch n Rechtecke der Breite $\Delta x = (b-a)/n$ und der Höhe $y_i = f(m_i)$ wobei m_i die i-te Streifenmitte ist.



$$F \approx \sum_{i=1}^{n} \left[f(m_i) \cdot \Delta x \right] = \Delta x \cdot \sum_{i=1}^{n} f(m_i)$$

Je grösser n gewählt wird, desto genauer ist die Näherung.

Programm zur Berechnung des bestimmten Integrals:

```
PROGRAM:INTEGRAL
:Input "LOWER: ",A
:Input "UPPER: ",B
:Input "N: ",N
:(B-A)/N→D
:0→S
:For(I,1,N)
:A+(I-0.5)*D→X
:prgmF
:S+Y→S
:End
:Disp D*S
```

Unterprogramm für die Funktion

```
PROGRAM:F
:X²+1→Y
```

Die Return-Anweisung

`Return` verlässt das Unterprogramm und kehrt zum aufrufenden Programm zurück, selbst wenn der Befehl in einer verschachtelten Schleife auftritt. Im Hauptprogramm hält Return die Ausführung an und der Hauptbildschirm erscheint.

Normalerweise benötigt man diese Anweisung nicht, denn ein impliziertes Return steht am Ende eines jeden als Unterprogramm verwendeten Programms.

Stop

Stop hält die Ausführung eines Programms an und zeigt wieder den Hauptbildschirm an. Stop kann am Ende eines Programms optional stehen.

5.1 Aufgaben

Aufgabe 5.1

Gegeben sind die beiden Programme `prgmA` und `prgmB`. Was gibt `prgmA` aus?

```
PROGRAM:A
:5→A
:1→B
:prgmB
:Disp A,B
```

```
PROGRAM:B
:A→T
:B→A
:T→B
```

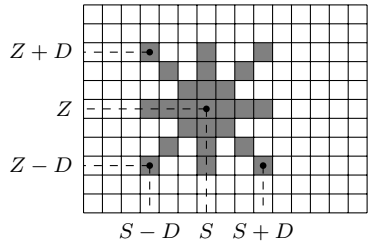
Aufgabe 5.2

Schreibe das folgende Programm `STARSKY` ab, das den Benutzer auffordert, eine Zahl N einzugeben. In der darauf folgenden For-Schleife wird N Mal

- eine Zufallszahl D von 0 bis 4 (inklusive) erzeugt,
- die Zufallskoordinaten Z (Zeile) bzw. S (Spalte) generiert, die einen Mindestabstand von D Pixeln vom Bildschirmrand haben,
- das Unterprogramm `STAR` aufgerufen, das einen Stern zeichnet.

```
PROGRAM:STARSKY
:PROMPT N
:ClrDraw
:For(J,1,N)
:randInt(0,4)→D
:randInt(D,164-D)→Z    [TI-84 Plus: 62 statt 164]
:randInt(D,264-D)→S    [TI-84 Plus: 94 statt 264]
:prgmStar
:End
```

Schreibe ein Unterprogramm `STAR`, das mit einer For-Schleife, in der die Variable `K` von `-D` bis `D` läuft, punktweise eine horizontale, eine vertikale und zwei diagonale Strecken aus je (2D+1) Pixeln zeichnet. Dabei ist der Punkt (Z,S) das Zentrum dieser Linien.



Benutze den Befehl `Pxl-On(<zeile>,<spalte>)` aus dem `draw`-Menü. Wer einen TI-84 Plus CE-T hat, kann auch `Pxl-On(<zeile>,<spalte>,<farbe>)` verwenden, wobei, `<farbe>`, eine ganze Zahl zwischen 10 und 19 ist.

Unterprogramm zum Zeichnen eines Sterns:

```
PROGRAM:STAR
:For(K,-D,D)
:Pxl-On(Z+K,S)
:Pxl-On(Z,S+K)
:Pxl-On(Z+K,S+K)
:Pxl-On(Z-K,S+K)
:End
```

Vor dem Starten von `STARSKY` sollte man im Funktionseditor allfällige Funktionen deaktivieren sowie unter [2nd] [zoom] [format] das Koordinatensystem mit `Axes: Off` ausschalten.

(Später nicht vergessen, die Achsen wieder zu aktivieren!)

6 Listen

Datentypen

Der TI-84 Plus verfügt über die zwei *elementaren Datentypen*:

- Zahlen
- Zeichenketten

Darüber hinaus gibt es den Datentyp einer *Liste*. Eine Liste ist ein *Datenstruktur*, die mehrere (numerische) Werte in einer bestimmten Reihenfolge zusammenfasst. Aus diesem Grund spricht man auch von einem *zusammengesetzten Datentyp*.

Speichern von Listen

Dafür verwendet man entweder die vorgegebenen Listenvariablen L_1 bis L_6 oder eine maximal 4 Zeichen lange Bezeichnung aus *Buchstaben*:

Beispiel: {2,3,5,7}→ʟA

Der Taschenrechner speichert die Liste jedoch nicht unter dem Namen A ab sonder unter dem Listennamen ʟA. Um die Liste wieder aus dem Speicher zu holen wählt man entweder den entsprechenden Listennamen aus dem LIST/NAMES-Menu aus oder gibt den Listennamen direkt an, wobei ihm das ʟ (ganz unten im Menu LIST/OPS) voranzustellen ist.

Die Grösse (Dimension) einer Liste bestimmen

dim(L_1) liefert die Anzahl der Elemente der Liste L_1 zurück.

Eine neue Liste erstellen

Mit 10→dim(L_1) wird eine Liste mit Platz für 10 Elemente erstellt und als Liste L_1 gespeichert. Die Elemente der Liste werden wie folgt bestimmt:

- Falls es die Liste noch nicht gibt, sind alle Einträge null.
- Falls die Liste bereits existierte und sie mehr als 10 Elemente hatte, werden die überzähligen Elemente „abgeschnitten“.
- Falls die Liste bereits existierte und sie 10 oder weniger Elemente hatte, werden fehlende Elemente durch Nullen ergänzt.

Eine Liste mit einer Zahl füllen

Wie wir oben gesehen haben, kann man sich bei einer neuen Liste nicht immer sicher sein, dass sie aus lauter Nullen besteht.

Mit Fill(0,L_1) wird jedes Element der Liste L_1 mit einer 0 überschrieben. Anstelle von 0 ist natürlich auch jede andere Zahl erlaubt.

Zugriff auf Listenelemente

Wer ein bestimmtes Listenelement zugreifen will, gibt die Position des Listenelementes in runden Klammern unmittelbar nach dem Listennamen an. Um beispielsweise den 7. Wert in der Liste L_2 zu erhalten, gibt man $L_1(7)$ ein. Umgekehrt kann man mit $42 \rightarrow L_1(7)$ die Zahl 42 an der 7. Stelle in der Liste L_1 speichern.

6.1 Aufgaben

Aufgabe 6.1

Schreibe ein Programm `LISTSUM`, das die Summe der Elemente der Liste L_1 mit Hilfe einer geeigneten Schleifenanweisung bestimmt und diese am Ende des Programms ausgibt.

Hinweis: Der Taschenrechner TI-84 Plus stellt uns die Listenfunktion `sum(...)` zur Verfügung. In dieser Aufgabe geht es aber darum die Summe mit Hilfe einer Schleife zu berechnen.

```
PROGRAM:LISTSUM
:0→S
:For(K,1,dim(L1))
:S+L1(K)→S
:End
:Disp "SUMME: ",S
```

Aufgabe 6.2

Schreibe ein Programm mit dem Namen `SQUARES`, das die Quadrate der ersten 100 natürlichen Zahlen (1^2, 2^2, ..., 100^2) in die Liste L_1 schreibt. Am Ende soll die Liste ausgegeben werden.

```
PROGRAM:SQUARES
:100→N
:N→dim(L1)
:For(K,1,N)
:K*K→L1(K)
:End
:Disp L1
```

Aufgabe 6.3

Schreibe ein Programm `LISTSUM2`, das alle positiven Zahlen der Liste L_1 addiert. (Vergleiche mit Aufgabe 6.1)

```
PROGRAM:LISTSUM2
:0→S
:For(K,1,dim(L₁))
:If L₁(K)>0
:S+L₁(K)→S
:End
:Disp "SUMME: ",S
```

Aufgabe 6.4

Schreibe ein Programm mit dem Namen `FIBO`, das den Benutzer nach einer natürlichen Zahl `N` fragt und die ersten `N` Fibonacchi-Zahlen in die Liste L_1 schreibt. *Hinweis:* die Folge der Fibonacchi-Zahlen besteht aus den zwei Startwerten $L(1) = 1$ und $L(2) = 1$. Jedes weitere Folgeglied ist die Summe seiner beiden Vorgänger.

$L(3) = L(1) + L(2) = 1 + 1 = 2$

$L(4) = L(2) + L(3) = 1 + 2 = 3$

$L(5) = L(3) + L(4) = 2 + 3 = 5$

usw.

```
PROGRAM:FIBO
:Prompt(N)
:max(N,2)→N
:N→dim(L₁)
:1→L₁(1)
:1→L₁(2)
:For(K,3,N)
:L₁(K-2)+L₁(K-1)→L₁(K)
:End
:Disp L₁
```

Kurzreferenz

Dieses Merkblatt ist unvollständig. Ausführlichere Informationen findet man im Benutzerhandbuch des TI-84 Plus, das man kostenlos von der Webseite

`https://education.ti.com/de/schweiz/guidebook/search`

herunterladen kann. Unter „Produkt“ gibt man die Kategorie *Grafiktaschenrechner* an und nach einem Klick auf *Anzeigen* erhält man eine Auswahl von Handbüchern, wo der Eintrag *TI-84 Plus / TI-84 Plus Silver Edition Guidebook* auszuwählen ist. Für den TI-84 Plus CE-T gibt es (noch) kein eignes Guidebook [Stand: 11.12.2016].

Erläuterungen

- TI-84-Basic-Schlüsselwörter sind in nicht proportionaler Schrift dargestellt. (`Prompt`)
- Variable Eingaben sind kursiv dargestellt (*`Bedingung`*)
- Eckige Klammern ([...]) bezeichnen optionale Eingaben. Diese Klammern müssen nicht eingegeben werden.

Programme verwalten

Ein neues Programm erstellen

1. [prgm] `NEW  1:Create New` [enter]
2. Programmnamen eingeben
3. Das Programm im Programmeditor schreiben. (→ Editierfunktionen)
4. Den Programmeditor mit [2nd] `quit` verlassen. Ein Programm ist immer im aktuellen Zustand gespeichert.

Ein Programm ausführen

1. [prgm] `EXEC`
2. Das Programm mit den Cursortasten auswählen.
3. Mit [enter] in den Hauptbildschirm einfügen.
4. Das Programm mit [enter] starten.

Ein bestehendes Programm bearbeiten

1. [prgm] `EDIT`
2. Das Programm mit den Cursortasten auswählen.
3. Mit [enter] das Programm in den Programmeditor laden.

Editierfunktionen

- clear: Löscht man den Inhalt der *gesamten Zeile*, auf der sich der Cursor befindet oder schliesst ein Menü, das innerhalb des Programm-Editors geöffnet wurde.
- del: Löscht das Zeichen (oder eine Zeilenschaltung) an der Position des Cursors.
- 2nd `[ins]`: Einfügen von Zeichen oder Leerzeilen (mit enter).

Ein Programm löschen

1. 2nd `mem`
2. `2:Mem Mgmt/Del...`
3. Den Eintrag `7:Prgm...` auswählen.
4. Den Cursor auf das zu löschenden Programm setzen.
5. Mit del das Löschen starten und bestätigen.

Ein Programm einfügen/kopieren

1. Ein neues oder bestehendes Programm öffnen (siehe oben).
2. 2nd `[rcl]` prgm `EXEC`
3. Das einzufügenden bzw. zu kopierende Programms auswählen und mit enter bestätigen
4. Den Einfügevorgang mit enter starten.

Ein- und Ausgabe (Menü `I/O`)

Input

`Input "`*`Text`*`",`*`Variable`*

Zeigt während der Programmausführung als Eingabeaufforderung einen *`Text`* aus maximal 16 Zeichen an. Danach ist eine Wert oder ein Ausdruck einzugeben und `ENTER` zu drücken. Der Wert wird in der *`Variablen`* gespeichert und die Programmausführung fortgesetzt.

Prompt

`Prompt` *`Variable1`*`[,`*`Variable2`*`,...]`

Bei Ausführung eines Programms zeigt `Prompt` alle Variablen nacheinander, gefolgt von einem Fragezeichen an. Nach jeder Eingabeaufforderung für die betreffende Variable ist ein Wert oder Ausdruck einzugeben und `ENTER` zu drücken. Die Werte werden unter den entsprechenden Variablennamen gespeichert und das Programm fortgesetzt.

Disp

`Disp` *`Wert1`* [,*`Wert2`*,*`Wert3`*,...]

- Ist *`Wert`* eine Variable, so wird der aktuelle Wert angezeigt
- Ist *`Wert`* ein Ausdruck, so wird dieser berechnet und das Ergebnis rechts in der nächsten Zeile angezeigt.
- Ist *`Wert`* ein in Anführungszeichen gesetzter Text, erscheint dieser links in der aktuellen Anzeigezeile.

Steuerbefehle (Menü `CTL`)

If

`If` *`Bedingung`*

Ist *`Bedingung`* wahr, so wird die nächste Anweisung ausgeführt. Ist *`Bedingung`* falsch, dann wird die unmittelbar danach folgende Anweisung übersprungen. Vergleichsoperatoren und logische Operatoren findet man im test-Menü.

Achtung: `If` führt bei unsachgemässer Anwendung schnell zu Fehlern.

If Then

`If` *`Bedingung`*
`Then`
`Anweisung(en)`
`End`

Folgt `Then` auf `If`, so wird eine Gruppe von Anweisungen ausgeführt, wenn die Bedingung wahr ist. `End` markiert das Ende der Anweisungsgruppe. `If`-Anweisungen können verschachtelt werden.

If Then Else

`If` *`Bedingung`*
`Then`
`Anweisung(en)`
`Else`
`Anweisung(en)`
`End`

Folgt `Else` auf `If ... Then`, so wird eine Gruppe von Anweisungen ausgeführt, wenn die Bedingung falsch ist. `If`-Anweisungen können verschachtelt werden.

For

`For(`*`Variable`*`,`*`Anfang`*`,`*`Ende`*`[,`*`Schrittweite`*`])`
`Anweisung(en)`
`End`

Führt die Anweisung(en) für die angegebene Variable von einem Anfangswert schrittweise fort, solange die Variable nicht grösser als der Endwert ist. Ohne Angabe einer Schrittweite hat diese den Wert 1. Eine Schrittweite kann auch negativ sein, wenn *`Anfang`*>*`Ende`* gilt. `For`-Schleifen können verschachtelt werden.

While

`While` *`Bedingung`*
`Anweisung(en)`
`End`

So lange *`Bedingung`* wahr ist, wird die Anweisungsgruppe ausgeführt. Andernfalls fährt das Programm mit den Anweisungen nach dem `End` weiter. `While`-Anweisungen können verschachtelt werden.

Repeat

`Repeat` *`Bedingung`*
`Anweisung(en)`
`End`

Führt die Anweisung(en) aus, so lange die *`Bedingung`* falsch ist. Beachte, dass die Bedingung *nach* der Ausführung der Anweisung(en) geprüft wird. Andernfalls fährt das Programm mit den Anweisungen nach dem `End` weiter. `Repeat`-Schleifen können verschachtelt werden.

Pause

`Pause`

Unterbricht die Ausführung des Programms, so dass man sich Ergebnisse ansehen kann. Während der Pause bewegt sich die Programmanzeige in der oberen rechten Bildschirmecke. Mit `ENTER` wird die Ausführung des Programms fortgestzt.

prgm

`prgm`*`Programmname`*

Damit werden andere Programme als Unterprogramme ausgeführt. Die Verwendung von `prgm` entspricht der Auswahl bestehender Programme aus dem `PRGM EXEC`-Menü. Mit `prgm` können auch Programme eingefügt werden, die noch nicht geschrieben sind.