RedCrab

Math V

Programmierer Manual

Version 5.0

Inhalt

1.1 RedCrab Program Interpreter
1.2 Kommentare
1.3 Identifikatoren
1.4 Gültigkeitsbereich eines Identifikators
1.5 String Konstante
1.6 Programmvariable
1.7 Numerische Ausdrücke
1.8 Boolesche Ausdrücke
1.9 Ausdrücke
1.10 Mehrzeilige Anweisungen

2.1 Program
2.2 Define
2.3 Let
2.4 While do
2.5 If Then
2.6 Else
2.7 Elseif
2.8 Function
2.9 Forward
2.10 Call
2.11 Result
2.12 Next
2.13 Index

1.1 RedCrab Program Interpreter

Diese Anleitung beschreibt die Syntax des *RedCrab* Interpreters. *RedCrab* verwendet eine eigene einfache Programmiersprache. Auch Benutzern ohne Programmier-Kenntnisse sollen in der Lage sein nach kurzer Einarbeitung einfache Funktionen zu schreiben.

Analog zum Arbeitsblatt wird bei reservierten Namen (Systemfunktionen und Anweisungen) nicht zwischen Groß-und Kleinschreibung unterschieden. Bei selbst-definierten Funktionen und Variablen wird Groß- und Kleinschreibung berücksichtigt.

Ein Programm besteht aus einer Folge von Anweisungen. Jede Anweisung beginnt mit einem Schlüsselwort. Das Zeilenende (*Linefeed*) beendet eine Anweisung. Die Schlüsselwörter *function*, *if* und *while* gelten für alle folgenden Anweisungen, bis der Anweisungsblock mit dem Schlüsselword *end* beendet wird. Extra Leerzeichen, Tabulatoren, Linefeeds und Kommentare werden vom Programm ignoriert.

Beispiel:
```
let a = 12
let b = 22
```

Es können mehrere Anweisungen in eine Zeile geschrieben werden, wenn sie mit einem Doppelpunkt getrennt werden.

Beispiel:
```
let a = 123 : let b = 22
```

1.2 Kommentare

Aus Gründen der Übersicht und zur Beschreibung der Funktionen, wird empfohlen, dass Sie Ihren Code dokumentieren, indem Sie Kommentare einfügen. Sie können auch mit Kommentar Symbole während der Programmentwicklung Teile des Programms deaktivieren ohne sie zu löschen. Kommentare können auf zweierlei Weise eingefügt werden:

- Ein Kommentar kann mit /* beginnen und mit den Zeichen */ enden.

- Sie können einen Kommentar mit einem doppelter Schrägstrich einleiten / /. Der Kommentar endet mit dem Ende der Zeile.

  Beispiel:
  ```
  /* Kommentar */
  // Kommentar
  ```

1.3 Identifikatoren

RedCrab Programme enthalten Verweise auf Module, Funktionen, lokale und globale Variablen und Konstanten. Mit Ausnahme von Konstanten, wird jeder Verweis über seinen Name identifiziert. Der Name besteht aus eine Folge von Buchstaben, Ziffern und Unterstrichen. Das erste Zeichen muss ein Buchstabe oder eine Unterstrich sein.

1.4 Gültigkeitsbereich eines Identifikators

Lokale Variablen werden innerhalb einer Funktion definiert. Auf Sie kann nicht von außerhalb ihrer Funktion zugegriffen werden.

Globale Variablen werden außerhalb einer Funktion definiert. Auf sie kann von allen Funktionen im Modul zugegriffen werden. Von externen Modulen und vom Arbeitsblatt können sie nur ausgelesen werden.

Funktionen können von allen Modulen und vom Arbeitsblatt aufgerufen werden.

1.5 String Konstante

String Konstante sind Sequenzen von Zeichen die in Anführungszeichen eingeschloßen sind. Strings müssen in einer Zeile geschrieben werden. Mit dem Punkt (.) können einzelne String Konstante miteinander verbunden werden.

Beispiel:
```
let s = "Hallo "
let t = s ."Welt"
```

Die Variable *t* enthält den Text: „*Hallo Welt*“

1.6 Programmvariable

Programmvariable müssen definiert werden , bevor sie verwendet werden. Dazu wird das Schlüsselword *define* verwendet. Optional kann in der Definition auch ein Wert zugewiesen werden. Für weitere Informationen lesen Sie unten die Beschreibung zu *define*.

1.7 Numerische Ausdrücke

Ein numerischer Ausdruck besteht aus einer Konstante, Variable, Zelle ein Datenfeldes oder einer Funktion die einen Zahlenwert liefert, oder mehrere davon die durch folgende arithmetische Operatoren verbunden sind:

*	Multiplikation
/	Division
Mod	Modulo
Div	Integer Division
+	Addition
-	Subtraktion

1.8 Boolesche Ausdrücke

Ein boolescher Wert bewertet einen Ausdruck als wahr (*TRUE*) oder unwahr (*FALSE*) und hat folgendes Format:

Ausdruck Operator Ausdruck

Die Ausdrücke können numerisch oder Text Strings sein. Wenn ein numerischer Ausdruck mit einem String verglichen wird der eine Zahl enthält, wird zum Vergleich der Wert der Zahl verwendet. Werden zwei Strings miteinander verglichen, werden sie immer als Strings behandet, unabhängig davon, ob sie Text oder Zahlen beinhalten. Die Operatoren zeigt die folgende Liste.

Operator	**Operation**
==	Gleich
< >	Ungleich
>	Größer als
>=	Größer oder gleich.
<	Kleiner als
<=	Kleiner oder gleich

Boolesche Ausdrücke können mit den *UND(&)* und *ODER(|)* Operatoren zusammengefaßt werden.

Beispiel:
```
(a >= b) & (c <= d)
(a == d) |  c
```

1.9 Ausdrücke

Für *if* und *while*-Anweisungen ist TRUE jede Zahl ungleich Null und FALSE ist gleich Null. Deshalb können Sie immer einen numerischen Ausdruck einsetzen, wo ein boolescher Ausdruck gefragt ist. Sie können ebenso einen booleschen Ausdruck verwenden, wo ein numerischer Ausdruck erwartet wird, was als 1 oder 0 inerpretiert wird. Sie können einen String-Ausdruck, der eine Zahl darstellt einsetzen wo ein numerischer Ausdruck erlaubt ist.

1.10 Mehrzeilige Anweisungen

Eine Feld-Definition kann in der nächsten Zeile fortgesetzt werden. Die aktuelle Zeile muß mit einem Komma oder einem Semikolon beendet werden.

Ein Statement kann in der folgenden Zeile fortgesetzt werden, wenn die aktuelle Zeile mit einem Backslash (\ ) beendet wird.

Eine lange Zahl kann in der folgenden Zeile fortgesetzt werden, wenn die aktuelle Zeile mit einem doppelten Backslash (\\) beendet wird.

Die folgende Tabelle zeigt einige Beispiele zu mehrzeiligen Statements:

Zweizeilige Statements	Interpretation
`Let m = [1,2,3;` `  4,5,6]`	`Let m = [1,2,3;4,5,6]`
`Let m = [1,2,3,4,` `  5,6,7,8]`	`Let m = [1,2,3,4,5,6,7,8]`
`Let v \` `  = 1 + 2`	`Let v = 1 + 2`
`Let v = 12345\\` `  6789`	`Let v = 123456789`
`Let s = "hello " \` `  ."world"`	`Let s = "hello " ."world"`

2.1 Programm

Ein Programm beginnt immer mit dem Schlüsselword *program* und dem Name der Programmbox.

Beispiel: `program name`

In dem folgenden Beispiel wird aus dem Arbeitsblatt die Funktion *root* des Programms *m1*aufgerufen.

```
m1.root(9)=3
```

```
program m1

function root(a)
  result =  sqrt(a)
end
```

2.2 Define

Die *define* Anweisung deklariert den Namen einer Variaben und weist ihr optional einen Wert zu. Wenn kein Wert zugewiesen wird, ist der Wert der Variable Null. Eine Variable kann nur verwendet werden, wenn der Name zuvor per *define* Anweisung oder in der Parameterliste einer Funktion deklariert wurde.

Die Syntax der *define* Anwisung ist:

```
define Name
define Name = Value
```

Als Wert kann eine Zahl, eine Valiable oder ein mathematischer Ausdruck eingesetzt werden. Das folgende Beispiel definiert die Variable ***x*** als ein leeres Datenfeld der Größe 20 * 8 (Zeilen,Spalten).

Beispiel : `define x[] = [1..20] * [1..8] fill 0`

2.3 Let

Let weist einer Variable einen Ausdruck zu. Die Syntax der *Let* Anweisung ist:

```
let variable = Ausdruck
```

Der Ausdruck bestehen aus einer oder mehreren Konstanten, Variablen oder Zellen eines Datenfeldes. Der Wert kann eine numerischer oder boolscher Ausdruck oder eine Text String sein.

Beispiel:

```
let x = 12
let x = y
let x = (12 + y) * z
let x = sin(45)
let x = "hello"
let x[5] = 16
```

Im letzten Beispiel wird dem Index [5] der Feldvariablen ***x*** der Wert 16 zugewiesen. Index [5] ist das fünfte Element des Datenfeldes. Beachten Sie, daß das Datenfeld mit Index [1] beginnt, im Gegensatz zu verschiedenen anderen Programmiersprachen, in den das erste Feld mit [0] indiziert wird.

2.4 While do

Mit *While* wird die Folge von Anweisungen (statements) zwischen *do* und *end* wiederholt, solange die Aussage der Bedingung (expression) wahr, bzw. ungleich Null ist. Wenn die Aussage falsch bzw. Null ist, wird das Program mit der Anweisung nach *End* fortgeführt.

Die Syntax der *While do* Anweisung ist:

```
while expression do
  statements....
end
```

Beispiel:

```
let i = 0
while i < 100 do
  Statement Sequence....
  let i = i + 1
end
```

2.5 If Then

If Then bietet die Möglichkeit einer bedingten Programmverzweigung. Die Anweisungen (statements) zwischen *then* und *end* werden nur ausgeführt, wenn der Ausdruck (expression) einen Wert ungleich Null (*TRUE*) liefert. Sonst wird der Anweisungsblock übersprungen und das Program mit der Anweisung nach *end* fortgesetzt.

Die Syntax der *if then* Anweisung ist:

```
if expression then
     statements....
end
```

2.6 Else

Die *else* Anweisung ist eine Erweiterung der *if then* Anweisung. Die Syntax der *if then else* Anweisung ist:

```
if expression then
     statements....
else
     statements.
end
```

Wenn der Ausdruck zwischen *if* und *then* Null (*FALSE*) ergibt werden die Anweisungen (statements) zwischen *then* und *else* übersprungen und statt dessen die Anweisungen zwischen *else* und *end* ausgeführt. Wenn der Ausdruck zwischen

if und *then* ungleich Null (*TRUE*) ergibt wird der Block zwischen *if* und *then* ausgeführt und der Block zwischen *else* und *end* übersprungen.

2.7 Elseif

Mit der Anweisung *elseif* können weitere Bedingungen für eine Programmverzweigung programmiert werden. Der auf *elseif* folgende Ausdruck wird nur ausgewertet, wenn die vorherigen *if* und *elseif* Ausdrücke den Wert Null (*FALSE*) ergeben. Wenn der Ausdruck einen Wert ungleich Null (*True*) ergibt, werden die auf *then* folgenden Anweisungen (statements) bis zur nächsten *elseif* oder *else* Anweisung ausgeführt. Dann wird das Programm nach *end* ausgeführt.

Die Syntax der *Elseif* Anweisung ist:

```
if expression then
  statements....
elseif
  statements.
elseif
  statements.
else
  statements.
end
```

2.8 Function

Die *function* Anweisung definiert eine Funktion. Eine Funktion ist ein benannter Block von Anweisungen. Sie kann vom Arbeitsblatt, aus anderen Funktionen des Programms, oder aus jedem anderen Programm-Modul namentlich aufgerufen werden. Eine Funktion liefert als Resultat einen einzelnen Wert, ein Datenfeld oder einen Text-String.

Die Syntax der *function* Anweisung ist:

```
function Name  (argument, argument......)
  statements...
end
```

Wenn keine Argumente an eine Funktion übergeben werden, muß in der Deklaration und im Funktionsaufruf eine leere Klammer hinter dem Namen stehen.

Beispiel: `function Name()`

2.9 Forward

In einem Programm können Funktionen nur aufgerufen werden, wenn sie vorher declariert sind. Der Zweck einer *forward*-Deklaration ist es, eine Funktion zu deklarieren die erst weiter unten im Quellcode implementiert ist. Dadurch können andere Funktionen, die vorwärts-deklarierte Routine aufrufen, bevor sie tatsächlich definiert ist. Dieses ist zum Beispiel notwendig, wenn Funktionen sich gegenseitig rekursiv aufrufen.

Die Syntax der *forward*-Anweisung lautet:

```
forward Name
```

2.10 Call

Die *call*-Anweisung ruft eine Funktion auf, ohne ein Ergebnis auszuwerten.

Die Syntax der *call*-Anweisung lautet:

```
Call FunctionName (argument, argument...)
```

2.11 Result

Die *result* Anweisung gibt einen Wert an die aufrufende Routine zurück. Der Rückgabewert kann eine Zahl, ein boolescher Wert, ein Text-String, ein Datenfeld oder das Resultat einer anderen Funktion sein. Es können konstante Werte oder Variable zugewiesen werden.

Die Syntax der *result* Anweisung lautet:

```
Result = Value
```

2.12 Next

Mit der ***Next*** Anweisung werden einzelnen Elementen von Datenfeldern Werte zugewiesen. Die Besonderheit von ***Next*** ist, daß kein Index angegeben wird. Der Index wird in der Feldvariablen selbst verwaltet. Mit jeder ***Next*** Zuweisung wird der interne Index um +1 inkrementiert.

Beispiel:

```
define x = [1..20]

next x
next x = 23
next x = 5.6
```

In dem Beispiel oben wird der Index durch die Anweisung ***Next x*** initialisiert. Dann wird **23** an **x[0]** und **5.6** an **x[1]** zugewiesen.

Wenn die initialisierte Variable einer anderen Variable zugewiesen oder als Parameter an eine Funktion übergeben wird, wird der Index auch übernommen.

Eine weitere Besonderheit ist, daß bei der Verwendung von ***Next*** kein Fehler durch Bereichsüberschreitung auftreten kann, weil ***Next*** automatisch das Datenfeld verlängert, wenn der Index das Ende erreicht hat. Bei sehr großen Datenmengen ist es aber sinnvoller, das Datenfeld vor der Verwendung ausreichend groß zu dimensionieren, weil die nachträgliche Verlängerung des Feldes zusätzliche Rechenzeit in Anspruch nimmt. Bei kleinen Datenmengen bis zu einigen tausend Records ist das meistens unerheblich. In jedem Fall ist es wichtig, daß mindestens

eine Zeile des Datenfelds mit der erforderlichen Anzahl von Spalten und Dimensionen definiert wird.

Next prüft, ob die Daten in der Zuweisung mit dem Datenfeld kompatibel sind. Die Anzahl der Dimensionen der zuzuweisenden Daten muß um eins geringer sein als die Dimension des Datenfeldes. In dem Beispiel oben, wurde ein einfacher Wert an ein Element eine eindimesionalen Felds zugewiesen. In dem Beispiel unten, wird ein eindimensionales Feld an ein Element eines zweidimensionalen Felds zugewiesen.

Die Anzahl der Spalten kann voneinander abweichen. In dem Beispiel unten ist x als ein zweidimensionales Datenfeld mit 3 Spalten definiert. In der vierten Zeile wird ein einspaltiges Feld mit dem Wert 99 zugewiesen. Die restlichen Elemente werden mit Nullen gefüllt.

In der fünften Zeile wird ein vierspaltiges Feld zugewiesen. Dort wird das vierte Element ignoriert.

Beispiel:

```
define x[] = [1..6]*[1..3] fill 1
next x
next x = [10,20,30]
next x = [99]
next x = [22,33,44,55]
```

Inhalt von x:

10	20	30
99	0	0
22	33	44
1	1	1
1	1	1
1	1	1

2.13 Index

Index ist eine Funktion die als Resultat den aktuellen Index einer Variable liefert (Index der letzten Zuweisung per ***Next***).

Beispiel:

```
i = index(x)
```