Hallo, Ihr da draußen, hier kommt der zehnte Teil von „Icon auf dem Raspberry Pi“. Ich hoffe, Ihr konntet den letzten Teil genießen und allen Schritten folgen. Wenn ja, solltet Ihr in der Lage sein, jegliche Arten des Zeichenketten-Scans zu erzeugen, was eine sehr machtvolle Programmiertechnik darstellt. In dieser und der nächsten Ausgabe wollen wir einige andere Techniken kennenlernen, die in den meisten anderen Programmiersprachen nicht verfügbar sind. Diese Techniken ermöglichen es Dir, in Icon besseren Code zu entwickeln.

10 Ausdrucks-Auswertung, Typ-Umwandlung, Sortier-Techniken

Neue Befehle für die Zeile `[keywords]` in home/pi/.config/geany/filedefs/filetypes.Icon.conf:

```
copy sort sortf
```

10.1 Ausdrucks-Auswertung

10.1.1 Begrenzende Erzeugung

Manchmal passt es nicht, alle Ergebnisse zu verbrauchen, die ein Generator in der Lage ist zu liefern. Dies ist eine Gelegenheit, den Begrenzungsoperator \ gefolgt von der Anzahl der zu erzeugenden Ergebnisse, anzuwenden. Lasst uns ein paar Zeile Code eingeben, um zu sehen, wie begrenzende Erzeugung genutzt werden kann.

Listing: home/pi/icon9_51/bin/limit.icn

```
1  procedure suspend_test(n, lim)
2     suspend (1 to n) \ lim do
3     {  write(" up")}
4     suspend (n to 1 by -1) \ lim do
5     {  write(" down")}
6  end
7
8  procedure main()
9     every writes(suspend_test(15,5))
10 end
```

#	Erläuterung
2	\ ist der Begrenzungsoperator, was bedeutet, dass die Prozedur `suspend_test()` nur die Anzahl der Werte einstellen wird, die durch das Argument `lim` festgelegt ist.

10.1.2 Wiederholte Alternierung

Manchmal möchtest Du die gleichen Folgen generieren – wieder und immer wieder. Dann solltest Du den wiederholten Alternierungsoperator | gefolgt von einem Ausdruck, der Ergebnisfolgen erzeugt, anwenden. Lasst uns ein paar Zeilen Code bzgl. wiederholter Alternierung eingeben.

home/pi/icon9_51/bin/repeated_alternation.icn

```
1  procedure suspend_test()
2     every suspend |((0 to 2) ||
3                     (0 to 2)) \16
4  end
5
6  procedure main()
7     every write(suspend_test())
8  end
```

#	Erläuterung
2	| ist der wiederholte Alternierungsoperator gefolgt von zwei Generatoren, die 0, 1 oder 2 erzeugen. Die Ergebnisse werden verkettet (Verkettungsoperator ||), um eine 2-ZeichenZeichenkette zu ergeben. Falls das erste Zeichen 0 ist, wird das zweite Zeichen 0 sein. Das nächste mal 1 und schließlich 2, bevor das erste Zeichen hochgezählt werden wird. Nach der Erzeugung von 16 solcher Paare, hat der Generator keine weiteren Ergebnisse einzustellen.

10.2 Typumwandlung

Im Teil 6 von “Icon – Ich kann's” haben wir eine Tabelle gezeigt, die Umwandlungsroutinen enthält. Jetzt wollen wir bzgl. dieses Merkmals mehr ins Detail gehen.

10.2.1 Implizite Typumwandlung

`write()` bzw. `writes()` sind die am häufigsten angewendeten Funktionen, die die implizite Typumwandlung nutzen. Da der Datentyp `string` der einzige ist, den `write()` / `writes()` in der Lage ist anzuzeigen, wird jeder andere Datentyp intern in eine Zeichenkette umgewandelt.

Die Umwandlung einer Zeichenmenge in einen numerischen Datentyp führt zuerst eine Umwandlung in eine Zeichenkette durch, die dann in einen numerischen Datentyp umgewandelt wird.

Alle Funktionen, denen eine Zeichenkette anstelle des Datentyps `cset` übergeben wird, wandeln diesen Datentyp ebenso implizit um, z.B. `upto()`.

Beachte: Es gibt keine implizite Typumwandlung beim Vergleichen von Werten in `case`Konstrukten für Tabellenschlüssel.

Listing: home/pi/icon9_51/bin/type_conversion.icn

```
1   procedure main()
2      s := "-15"
3      n := -s
4      write(s, "\t", n)
5
6      a := 10
7      b := (a < "15")
8      write(name(b), " = \t", b, "\t(type = ",
9            type(b), ")\t", image(b))
10
11     a := "10"
12     b := (a << "15")
13     write(name(b), " = \t", b, "\t(type = ",
14           type(b), ") \t", image(b))
15
16     a := 10
17     b := (a << "15")
18     write(name(b), " = \t", b, "\t(type = ",
19           type(b), ") \t", image(b))
20
21     a := "10"
22     b := (a < "15")
23     write(name(b), " = \t", b, "\t(type = ",
24           type(b), ")\t", image(b))
25
26     write(.b, "\t+", a, "\t= ", b +:= a)
27  #  a := "z"
28  #  write(.b, "\t+", a, "\t= ", b +:= a)
29
30     a := 1
31     case a of
32     {1:  write(name(a), " is an integer")
33      "1":write(name(a), " is a string")
34      '1':write(name(a), " is a cset")
35     }
36
37     T := table()
38     T[1] := 1
39     T["1"] := 1
40     T['1'] := '1'
41     write("size of table T = ", *T)
42
43     s := "the Icon Programming Language
44              for the famous Raspberry Pi"
45     write(s)
46     write(cset(s))
47     write(string(cset(s)))
48
49     s := set([1,2,3,4,5,6,7,8,9])
50     t := ""
51     every t ||:= !s
52     write(t, "\tcset = ", cset(t),
53           "\tstring = ", string(cset(t)))
54
55     s := cset(['1','2','3','4','5','6',
56               '7', '8','9'])
57     t := ""
58     every t ||:= !s
59     write(t, "\tcset = ",
60           cset(t), "\tstring = ",
61           string(cset(t)))
62  end
```

#	Erläuterung
7 12 17 22	Der wichtigste Operator, der steuert, welche Art von Typumwandlung durchgeführt werden wird, ist der Vergleichsoperator. Denk an die Kapitel 5 und 6 zurück: Der Vergleichsoperator für numerische Werte ist z.B. <, der für Zeichenketten <<. Dieser Operator gibt den Datentyp an, der aus Vergleichsoperationen durch implizite Typumwandlung resultieren soll.

10.2.2 Explizite Typumwandlung

Es gibt fünf Funktionen, die für explizite Typumwandlungen genutzt werden können: `integer()`, `real()`, `numeric()`, `string()`, `cset()` (siehe Kapitel 6 für mehr Details).

Aber es gibt eine andere sehr nützliche Anwendung expliziter Typumwandlung. Stell Dir vor, Du hast eine Zeichenkette und bist an den Zeichen darin interessiert – nicht der Reihenfolge.

Üblicherweise wirst Du mehrere Schleifen programmieren. Die erste Schleife würdest Du brauchen, um die Zeichen zu sammeln, die dann vielleicht in einer Tabelle oder einer Liste gespeichert werden. Die nächste Schleife wird die Zeichenkette aufbauen, die aus den Zeichen besteht, die durch die erste Schleife gefunden wurden. Durch Nutzung von Icon's eingebauten Funktionen ist die Lösung sehr einfach: `s := string(cset(s))`. Dieser Ausdruck setzt auch alle Zeichen in die lexikalische Reihenfolge.

10.2.3 Werte vergleichen

In den Teilen 5 und 6 haben wir die Vergleichsoperatoren wie = für numerische Werte und == für Zeichenketten-Werte vorgestellt. Es gibt noch einen anderen Vergleichsoperator, ===, der vergleicht, was auch immer verglichen werden soll.

Lasst uns ein paar Zeilen Code eingeben.

Listing: home/pi/icon9_51/bin/type_comparing.icn

```
1  procedure main()
2      x := 1
3      y := 1
4      write("Integer ",
5                if (x = y) then "true"
6                           else "false")
7      write("General ",
8                if (x === y) then "true"
9                             else "false")
10
11     x := "1"
12     y := "1"
13     write("String ",
14               if (x == y) then "true"
15                           else "false")
16     write("General ",
17               if (x === y) then "true"
18                            else "false")
19
20     x := list(1,1)
21     y := list(1,1)
22     write("List ",
23               if (x === y) then "true"
24                            else "false")
25
26     x := y := list(1,1)
27     write("List ",
28               if (x === y) then "true"
29                            else "false")
30     write(x[1], " <-> ", y[1])
31     x[1] := 2
32     write("List ",
33               if (x === y) then "true"
34                            else "false")
35     write(x[1], " <-> ", y[1])
36
37     x := list(1,"List 1")
38     z := [x]
39     y := copy(z)
40     write("Copy ",
41               if (z === y) then "true"
42                            else "false")
43     write("Copy ",
44        if (z[1] === y[1]) then "true"
45                            else "false")
46 end
```

#	Erläuterung
5	Vergeichen numerischer Werte mit dem Operator =
8	Vergleichen numerischer Werte mit dem Operator ===
14	Vergleichen von Zeichenketten mit dem Operator ==
17	Vergleichen von Zeichenketten mit dem Operator ===
23 28 33	Vergleichen von Listen oder Listenelementen mit dem Operator ===. Der erste Vergleich scheitert, weil die Listen nicht identisch sind, obwohl sie in Größe und Inhalt gleichwertig sind. Der zweite Vergleich ist erfolgreich, weil die Listenzuweisung (Zeile 26) Strukturen nicht kopiert und die Argumente identische Werte haben. Der dritte Vergleich ist das gleiche, aber das Element der Liste `x` wurde geändert. Da sich beide Listen im gleichen Speicherbereich befinden, wirkt sich die Zuweisung in Zeile 31 ebenso auf die Elemente der Liste `y` aus – daher hat der Vergleich Erfolg.
38	Macht eine neue Liste
39	**Syntax: copy(x1):x2 kopiert Wert** `copy(x1)` erzeugt eine Kopie von `x1`, falls `x1` eine Struktur ist; anderenfalls erzeugt es `x1`. Diese Kopie unterscheidet sich vom Original, obwohl die Größen von `x1`, `x2` und ihre Inhalte identisch sind.
41	Weil die Liste `y` (eine Kopie von `z`) sich von `z` unterscheidet, scheitert der Vergleich.
44	Aber Vergleichen der Elemente ist erfolgreich

10.2.4 Werte kopieren

Jeder Wert kann mittels der Funktion `copy(x)` kopiert werden. Eine neue Kopie wird gemacht, falls `x` ein strukturierter Datentyp ist (Liste, Tabelle, Menge, Zeichenmenge, Record). Aber die Kopie unterscheidet sich vom Original, obwohl sie hinsichtlich Größe und Inhalt identisch sind. Der Grund: Sie verwenden unterschiedliche Speicherbereiche. Siehe das Listing in 10.2.3.

10.3 Sortier-Techniken

10.3.1. Records, Listen und Mengen

Die Funktion `sort(x)` erzeugt eine Liste mit den Werten in sortierter Reihenfolge. Falls die Liste oder Menge vielerlei Datentypen enthält, werden die Werte zuerst nach dem Typ sortiert. Nach dem Sortieren liegen die Datentypen in folgender Reihenfolge vor:

- Null-Wert
- Ganzzahlen
- Dezimalzahlen
- Zeichenketten
- Zeichenmengen
- window
- file
- Co-Expressions
- Prozeduren, Funktionen und Record-Konstruktoren
- Listen
- Mengen
- Tabellen
- Record-Typen

Lasst uns ein paar Zeilen Code eingeben.

Listing: home/pi/icon9_51/bin/sorting_structures.icn

```
1  link graphics
2
3  record rec1(a,b)
4  record rec2(c,d)
5  record rec3(e,f)
6  record rec4(g,h)
7  record rec5(i,j)
8
9  procedure print(l)
10     every i := 1 to *l do
11        {  write(i, "\t", image(l[i]))
12        }
13 end
```

Listing: home/pi/icon9_51/bin/sorting_structures.icn (Fortsetzung)

```
14
15 procedure main()
16    L := list()
17    every i := 1 to 100 do
18    {  typ := ?["&null", "integer", "real", "string", "cset", "window", "file",
19                "co-expression", "procedure", "list", "set", "table", "record"]
20       case typ of
21       {     "&null":           {put(L, &null)}
22             "integer":          {put(L, ?100000)}
23             "real":             {put(L, 100000.0 *?0)}
24             "string":           {put(L, ?&letters || ?&letters || ?&letters || ?&letters ||
25                                         ?&letters || ?&letters || ?&letters || ?&letters ||
26                                         ?&letters || ?&letters)}
27             "cset":             {put(L, cset(?&ascii || ?&ascii || ?&ascii || ?&ascii ||
28                                              ?&ascii || ?&ascii || ?&ascii || ?&ascii ||
29                                              ?&ascii || ?&ascii))}
30             "window":           {put(L, w := WOpen()); WClose(w)}
31             "file":             {fh := open("test.abc", "w")
32                                  put(L, fh)
33                                  close(fh)
34                                  remove("test.abc")
35                                 }
36             "co-expression":    {put(L, (?3)(&main, &current, &source))}
37             "procedure":        {f := (?7("abs","exp","sin","cos","tan", "write", "writes")
38                                  put(L, proc(f, 0))
39                                 }
40             "list":             {put(L, list(?10))}
41             "set":              {put(L, set())}
42             "table":            {put(L, table())}
43             "record":           {put(L, (?5)(rec1(), rec2(), rec3(), rec4(), rec5()))}
44       }
45    }
46    print(L)
47    L := sort(L)
48    print(L)
49 end
```

#	Erläuterung
47	**Syntax: sort(X):X** **sortiert Struktur** `sort(X)` erzeugt eine Liste , die die Werte von `X` enthält. Falls `X` ein Record, Liste oder Menge ist, erzeugt `sort(X)` die Werte von `X` in sortierter Reihenfolge. Die Liste `L` wird sortiert, das Ergebnis wird `L` zugewiesen.

10.3.2 Tabellen

Das Sortieren von Tabellen unterscheidet sich ein wenig von dem anderer strukturierter Datentypen. Es gibt ein weiteres Argument, das benutzt werden kann, um die Form des Ergebnisses und die Sortierreihenfolge zu beinflussen.

```
T := sort(T, i)
```

Lasst uns ein paar Zeilen Code eingeben.

Listing: home/pi/icon9_51/bin/sort_table.icn

```
1  procedure print_table(x)
2     case type(x) of
3     {"table":{i := 0
4               every k := key(x) do
5               {i +:= 1
6                write(i, " => ", k,
7                      ":", x[k])
8               }
9              }
10     "list": {i := 1
11              while i < *x do
12              {case type(x[i]) of
13               {"list":{l := x[i]
14                        write(i, " =>\t",
15                              get(l),
16                              "\t:\t",
17                              get(l))
18                       i +:= 1
19                      }
20                default:{write(i, " => ",
21                               image(x[i]),
22                               ":\t",
23                               image(x[i+1]))
24                       i +:= 2
25                      }
26               }
27              }
28             }
29     }
30     write()
31  end
32
33  procedure main()
34     T := table()
35     L := list()
36
37     every i := 1 to 20 do
38     {  T[?1000] := ?100000
39     }
40     print_table(T)
41     L := sort(T,1); print_table(L)
42     L := sort(T,2); print_table(L)
43     L := sort(T,3); print_table(L)
44     L := sort(T,4); print_table(L)
45  end
```

#	Erläuterung
3-9	Zeigt die Tabelle gerade nach deren Erzeugung an
13-19	Zeigt die Liste an, die nach Sortieren mit i = 1 oder 2 erzeugt wurde. Diese Liste besteht aus Listen, die einen Schlüssel und einen Wert enthalten.

#	Erläuterung
20-25	Zeigt die Liste an, die nach dem Sortieren mit i = 3 oder 4 erzeugt wurde. Diese Liste sind abwechselnde Schlüssel / Wert-Paare.
41 42 43 44	**Syntax: sort(X, i):L sortiert Struktur** `sort(X, i)` erzeugt eine Liste, die die Werte von `X` enthält. Falls `X` eine Tabelle ist, erzeugt `sort(X, i)` eine Liste, die durch Sortieren der Elemente von `X` erhalten wurde, abhängig vom Wert `i`. Für `i` = 1 oder 2 sind die Listenelemente Zwei-Element-Listen von Schlüssel / Wert-Paaren. Für `i` = 3 oder 4 sind die Listenelemente abwechselnde Schlüssel und Werte. Sortieren nach Schlüsseln erfolgt bei ungeradem `i`, nach Werten bei geradem.

10.3.3 Nach Feldern sortieren

Listen, Records oder Mengen können nach dem Feldindex sortiert werden, sofern sie von der Funktion `sortf()` angewendet werden. Lasst uns ein paar Zeilen Code eingeben.

Listing: home/pi/icon9_51/bin/sortf.icn

```
1   record person(name, age, salary)
2
3   procedure print_list(l)
4      every i := 1 to *l do
5      {  every j := 1 to *l[1] do
6         {     writes(l[i,j], "\t")
7         }
8         write()
9      }
10     write()
11  end
12
13  procedure main()
14     L := [    ["abc", 55, 98700],
15               ["ijk", 29, 43000],
16               ["xyz", 43, 78000]]
17                     print_list(L)
18     L := sortf(L, 1); print_list(L)
19     L := sortf(L, 2); print_list(L)
20     L := sortf(L, 3); print_list(L)
21
22     write()
23     L := [    person("abc", 55, 98700),
24               person("ijk", 29, 43000),
25               person("xyz", 43, 78000)]
26                     print_list(L)
27     L := sortf(L, 1); print_list(L)
28     L := sortf(L, 2); print_list(L)
29     L := sortf(L, 3); print_list(L)
30  end
```

#	Erläuterung
2-8	Zeigt die Liste an
4-10	Zeigt die erzeugte Liste an

18,19 20 27,28 29	**Syntax: sortf(X, i):L** **Sortiert Struktur nach dem Feld** `sortf(X, i)` erzeugt eine sortierte Liste der Werte vom Record, Liste oder Menge `X`. Listen- und Record-Werte in `X` werden durch Vergleichen der Werte ihrer `i`-ten Felder sortiert. Der Wert von `i` kann negativ aber nicht 0, sein. Zwei Strukturwerte in `X`, die gleiche `i`-te Felder aufweisen, werden wie beim regulären Sortieren auch sortiert, aber Strukturen, denen es am `i`-ten Feld mangelt, erscheinen vor Strukturen, die dieses Feld haben.

Obwohl die Datei `sort.icn` innerhalb der IPL Funktionen hinsichtlich verbesserter Sortierung enthält, empfehle ich, diese Algorithmen selber zu programmieren, um die Schritte zu verstehen, wie Algorithmen in Code umgeformt werden – und um Erfahrungen im Programmieren, Testen und Validieren Deines Codes zu sammen.

Übungsaufgabe:

Versuche ein Programm zu entwickeln, dass für Büchereien benutzt werden kann: Gib Daten für den Autor, Buchtitel, Verlag, Veröffentlichungsjahr und einige Schlüsselworte ein. Erzeuge einen Record, der diese Daten hält, und eine Liste, die diese Records hält. Die Daten sollten in eine Datei geschrieben werden, bevor das Programm endet / unmittelbar nach Programmstart aus einer Datei gelesen werden. Wir wollen die Liste nach jedem Feld sortieren, um so unterschiedliche Listen zu erhalten. Das Programm soll die Suche innerhalb der Autoren, Buchtitel, Verlage, Veröffentlichungsjahre und Schlüsselworte ermöglichen.

Falls Du keine Bücher magst, kannst Du Ähnliches auch für Deine CD/DVD-Sammlung, für Deine elektronischen Bauteile oder für was immer Du möchtest, programmieren.

Das war's für heute! Genießt Euer Leben bis zum nächsten Mal!

Was kommt demnächst?

11 Zeichenkettennamen und Zeichenketten-Anrufung, Speichermanagement, Befehle ausführen, Verzeichnisse ändern, Umgebungsvariablen, Datum und Zeit, Icon-Identifizierung, Programm-Beendigung