Uebungsstunde vom 7.12.01, Info I

7. Dezember 2001

1 RegenModul

1.1 Design eines GUI (Graphical User Interface aufgrund der Aufgabenstellung von Serie 3, a) bis c)

1.2 Beispiellösung:

```
MODULE Rain;

IMPORT RandomNumbers, Out, In;

CONST MaxDays = 10000;

VAR days: ARRAY MaxDays OF REAL; (* Every day's rainamount *)
   accumulated: ARRAY MaxDays OF REAL; (* The accumulated amounts for ex. c *)
```



```
(* Ex 3a ...................................................................*)

(* Fill the array days with randomnumbers.
        Establish: accumulated[i] := days[0]+days[1]+...+days[i-1], i>0; *)

PROCEDURE InitArrays*;
VAR i: INTEGER;
        acc: REAL;
BEGIN
        acc := 0;

        FOR i := 0 TO MaxDays - 1 DO
                days[i] := RandomNumbers.Uniform();
                accumulated[i] := acc;
                acc := acc + days[i];
        END;
END InitArrays;


PROCEDURE CountRecords*;
VAR i, count: INTEGER;
        currentrecord: REAL;
BEGIN
        currentrecord := days[0]; (* The first element is always a record *)
        count := 1;

        FOR i := 1 TO MaxDays - 1 DO
                IF currentrecord < days[i] THEN
                        currentrecord := days[i]; (* adjust the new maximum *)
                        INC(count);
                END;
        END;

        Out.Int(count, 5);
        Out.String(" records.");
        Out.Ln;
END CountRecords;


(* Ex 3b .....................................................................*)

(* Calculate the theoretical value. See seperate sheet for more information *)
PROCEDURE TheoreticalValue*;
VAR i: INTEGER;
VAR value: REAL;

BEGIN
        value := 0;

        (* Decreasing loop variable to get higher numerical precision.
           Try increasing to see! *)
        FOR i := MaxDays TO 1 BY -1 DO
                value := value + 1.0/i; (* 1.0 is important to do computation in REAL.
```

```
                                  1 would do it in int *)
        END;

        Out.String("The expected value is ");
        Out.Real(value, 20);
        Out.String(" records.");
        Out.Ln;
END TheoreticalValue;


(* Ex 3c ..................................................................*)

(* Searching a certain sum "sum" from a day "since" is equivalent to searching
        the sum "sum+accumulated[since]" from day 0.
        Return value: The number of days in between.
*)
PROCEDURE RainSumSince(sum: REAL; since: INTEGER): INTEGER;
VAR i, j, m, result: INTEGER;

BEGIN
        sum := sum + accumulated[since];

        (* Do binary search *)
        i := -1;
        j := MaxDays;

        WHILE j # i+1 DO
                m := (i + j) DIV 2;
                IF accumulated[m] <= sum THEN
                        i := m;
                ELSE
                        j := m;
                END;
        END;

        (* Binary search has terminated. Has the exact sum been found? *)
        IF (i >= 0) & (accumulated[i] = sum) THEN
                result := i - since; (* yes, found *)
        ELSE
                result := j - since; (* no, not found, acc[j] is larger than sum *)
        END;

        RETURN result;
END RainSumSince;


PROCEDURE Inquiry*;
VAR sum: REAL;
        since, result: INTEGER;
BEGIN
        In.Open;
        In.Real(sum);
        In.Int(since);
```



```
        IF (~In.Done) THEN
                Out.String("Start with Rain.Inquiry amount startday"); Out.Ln;
        END;

        result := RainSumSince(sum, since);
        IF (result + since) = MaxDays THEN
                Out.String("not reached");
                Out.Ln;
        ELSE
                Out.Int(result, 5);
                Out.String(" day(s) in between.");
            Out.Ln;
        END;
END Inquiry;


(* Ex 3d ..................................................................*)

(* Insert a day in a sorted data structure.
        To avoid swapping through whole array, establish a second data
        structure which is a index chain.
        Eg:
        Values:      27    39   18    24
        Chain:         1    -1    3       0
        Entry point: 2.
        Go to the entry point (index 2, value 18.)
        The chain now points to index 3, value 24.
        The chain there points to index 0, value 27.
        The last (and largest) element points to -1 to indicate end of chain.
*)
PROCEDURE InsertDay(rainamount: REAL; day: INTEGER; VAR statdays: ARRAY OF REAL;
        VAR indexchain: ARRAY OF INTEGER; VAR first: INTEGER);
VAR previous, current: INTEGER;
BEGIN
        statdays[day] := rainamount;
        indexchain[day] := -1;

        previous := -1;
        current := first;

        WHILE (current > -1) & (statdays[current] < rainamount) DO
                previous := current;
                current := indexchain[current];
        END;

        IF previous > -1 THEN
                indexchain[day] := current;
                indexchain[previous] := day;
        ELSE
                IF first > -1 THEN
                        indexchain[day] := first;
                END;
                first := day;
        END;
```

```
END InsertDay;


PROCEDURE PrintDays(VAR statdays: ARRAY OF REAL;
        VAR indexchain: ARRAY OF INTEGER; first: INTEGER);
VAR i: INTEGER;
BEGIN
        i := first;
        WHILE (i > -1) DO
                Out.String("Amount: ");
                Out.Real(statdays[i], 20);
                Out.String(" on day: ");
                Out.Int(i, 3);
                Out.Ln;
                i := indexchain[i];
        END;
END PrintDays;


(* Adds days sorted to an array with chained indizes.
        For every day, generate a random rain amount and insert it into the list.
        Why I did not use the same array "days" as above:
        I wanted to avoid clashes when calling this routine between calls of the
        routines above.
*)
PROCEDURE AddDays*;
VAR i, first, number: INTEGER;
        r: REAL;
        indexchain: ARRAY MaxDays OF INTEGER;
        statdays: ARRAY MaxDays OF REAL;

BEGIN
        first := -1;
        In.Open;
        In.Int(number);

        IF (~In.Done) THEN
                Out.String("Start with Rain.AddDays NumberOfDays"); Out.Ln;
                RETURN;
        END;

        FOR i := 0 TO number - 1 DO
                r := RandomNumbers.Uniform();
                InsertDay(r, i, statdays, indexchain, first);
        END;

        PrintDays(statdays, indexchain, first);

END AddDays;


BEGIN
        RandomNumbers.InitSeed(314519);
        InitArrays;
```



```
END Rain.

(* Freeing the module *)
System.Free Rain~

(* Commandos for ex 3a *)
Rain.InitArrays~
Rain.CountRecords~

(* Commando for ex 3b *)
Rain.TheoreticalValue~


(* Commando for ex 3c *)
Rain.Inquiry 23.5 0~ (* When, from day 0, was the accumulated sum >= 23.5? *)
Rain.Inquiry 40.1 22~ (* When, from day 22, was the accumulated sum >= 40.1? *)
Rain.Inquiry 1000 9998~


(* Commando for ex 3d *)
Rain.AddDays 10~ (* To have fewer pairs *)
Rain.AddDays 30~
```

2 Vorbesprechung Serie 5

2.1 Decoder

Input: String -> Output: LONGINT

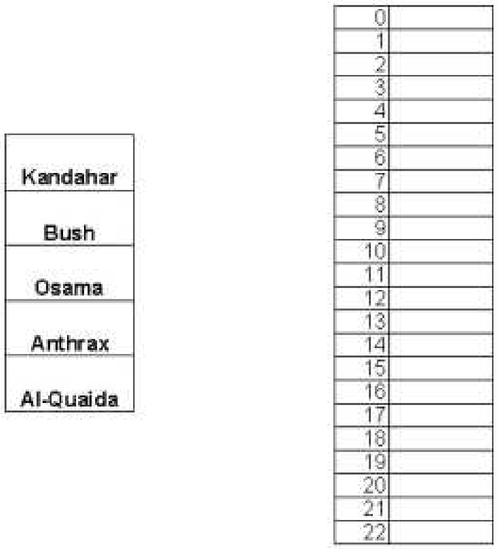
- "12345" $\rightarrow$12345
- "223x85" $\rightarrow$ -1
- "3520192749" $\rightarrow$ -1
- "0004150" $\rightarrow$ 4150
- "00000" $\rightarrow$ 0

2.2 Sortieren Dual zu Quicksort

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
3	9	13	31	41	42	68	81	2	6	7	15	19	44	48	72

2.3 Hashing

Idee: Effizient berechenbaren, eindeutigen Stellvertreter für komplexe Ausdrücke finden



mögliche Hashfunktionen:

- $c_i * b^i + c_{i-1} * b^{i-1} + ... + c_1 * b^1 + c_0 = (((...(c_i)*b + c_{i-1}) *b + ...)*b + c_1)*b + c_0 \pmod p$
- $(c_i * b * (i+1)) + ... + (c_1 * b * 2) + (c_0 * b * 1)$

Berechnung des ASCII-Werts: ORD(c)

Ziel: Kollisionsfreiheit $\rightarrow$p anpassen

2.4 Flugplan

Madrid	Dijon	800	896
Nizza	New York	315	910
Dakar	Amsterdam	1028	1360
Berlin	Chicago	280	710
Ljubliana	London	691	805
Moskau	New York	860	250

Diego	Indianapolis	150	398
Chicago	Orlando	890	1035
Las Vegas	Halifax	370	678
New York	Atlanta	1086	1210
New York	Detroit	450	678
Chicago	Austin	800	1283

Bedingungen:

- Connection Point festlegen
- 120 Minuten Umsteigezeit