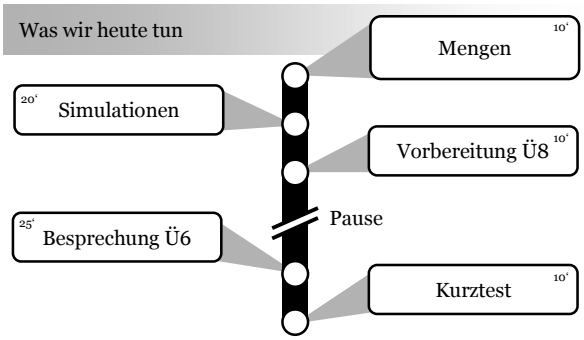
Informatik I WS 02/03

7. Januar 2003

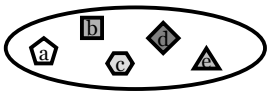
Übungsserie 7 jetzt abgeben
Neues Kapitel im Skript zum Thema *Warteschlangen*

Was wir heute tun

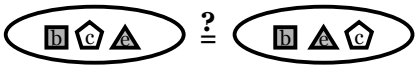


Was sind Mengen?

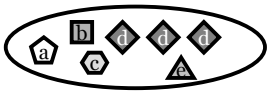
Mengen enthalten Elemente!



haben Mengen eine Reihenfolge?



können gleiche Elemente mehrmals vorkommen?



Arten von Mengen

Definitionssache!

	Reihenfolge	Mehrfachelemente
Bag	nein	ja
Set	nein	nein
Sequence	ja	ja

Vorbereitung Übungsserie 8 Aufgabe 1

Aufgabe 1. Reduktion von Bag auf Set

a. Entwickeln sie ein Modul **Store**, welches eine verkettete Liste zur Speicherung von Zahlen implementiert.
Store bietet die folgenden Operationen an:

- `Init();`
- `Append(v : INTEGER);`
- `Delete(v : INTEGER);`
- `Exists(v : INTEGER): BOOLEAN;`

} Ist das ein Bag oder ein Set?

b. Implementieren sie eine Prozedur, welche alle mehrfach vorkommenden Zahlen aus **Store** eliminiert.

Wie wandelt man einen Bag in einen Set um?

- jedes Element mit jedem vergleichen O(n!)
- Liste sortieren O(n log n)
- jedes Element hashen & Kollision prüfen O(n)

welches ist jetzt das Beste?

Was ist Simulation?

Simulation = Durchführung eines Vorganges in einem gewählten Modell

Echte Welt –
Auto fährt in einen Baum

Simulation –
mit dem Modell der Physik

Auto: v = 4 m/s
Material: Metall, stabil
Form: eckig, sperrig

Formel für Stoss:
$\frac{2m_2}{m_1+m_2}\vec{v}_2+\frac{m_1-m_2}{m_1+m_2}\vec{v}_1$

Baum: v = 0 m/s
Material: Holz, weich
Form: lang, rund

Simulieren auf dem Computer

Unsere Welt ist die Erde
Als Welt der Simulation nehmen wir Oberon

Auf der Erde werden Autos gebaut
In Oberon macht das eine Prozedur

```
PROCEDURE FabriziereAuto( G: Grösse; M: Material );
```

Auf der Erde wachsen Bäume

```
PROCEDURE PlatziereBaum( B: Baum; X, Y: LONGINT);
```

Auf der Erde wird herumgefahren

```
PROCEDURE PlatziereAuto( A: Auto; X, Y: LONGINT);
PROCEDURE BeschleunigeAuto( V: Vector );
```

Was passiert, wenn man auf einen Baum beschleunigt?
physikalisch ein gerader zentraler Stoss $\frac{2m_2}{m_1+m_2}\vec{v}_2+\frac{m_1-m_2}{m_1+m_2}\vec{v}_1$

```
2*M2 DIV (M1+M2) * V2 + (M1-M2) DIV (M1+M2) * V1;
```

Das Modell einer Simulation

Wie sich die Welt der Simulation verhalten soll, beschreibt das Modell

Wann aber läuft das Modell ab?

Bis jetzt sind nur Bäume & Autos plaziert!

Ausgangslage; eine Art Eingabe
Jetzt können wir also die virtuelle Welt laufen lassen!
Dazu braucht man eine Prozedur

```
PROCEDURE SimulationStarten()*;
```

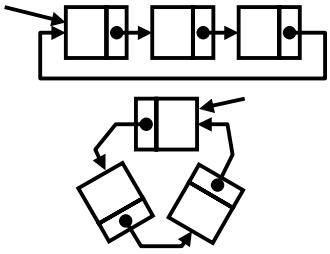
In dieser Prozedur ist das Modell umgesetzt:
- Autos fahren herum
- Bäume stehen herum
- Wenn ein Auto irgendwo durchfährt, wo ein Baum steht, gibt es einen Zusammenstoss

Notizen

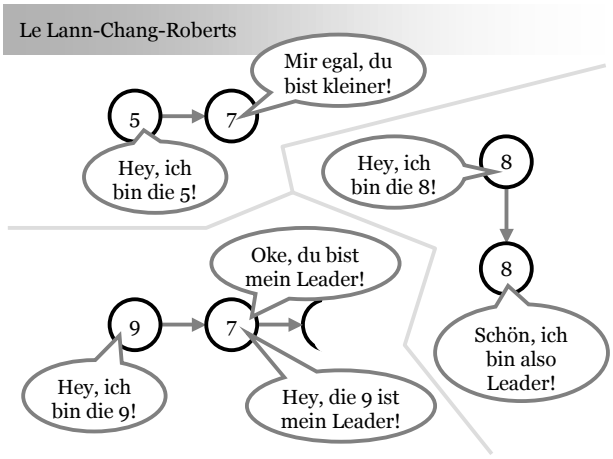
Vorbereitung Übungsserie 8 Aufgabe 2

Ein Ring, implementiert durch eine verkettete Liste, von N nummerierten Prozessen möchte einen Leader aus ihrem Kreis finden. Dabei gehen sie nach Le Lann-Chang-Roberts wie folgt vor.

Ring = Liste, deren Ende gleich dem Anfang ist:



Le Lann-Chang-Roberts



Simulation starten

Als Eingabe hat man die Liste
`PROCEDURE FuegeNeuesElementHinzu( Wert: LONGINT );`

Als Modell nehmen wir Le Lann-Chang-Roberts

Als Visualisierung drucken wir Information auf dem Bildschirm aus (Module `Out`)
Zeitmessung: `PROCEDURE Time(): LONGINT;`

Die Startprozedur:
`PROCEDURE StarteLeLannChangRoberts( L: Ringliste );`

hier drinnen ist der
Le Lann-ChangRoberts Algorithmus

Repetition Rekursion

Was ist Rekursion?

```
PROCEDURE Fakultaet(n : INTEGER): INTEGER;

BEGIN

      IF n <= 1 THEN
            RETURN 1;
      ELSE
            RETURN n * Fakultaet(n-1);
      END
END Fakultaet;
```

Repetition Rekursion

Was passiert bei der Rekursion?

Ablauf bei Aufruf `a := Fakultaet(❸);`

```
❸ IF n <= 1 THEN  (* n = 3 *)
  ELSE RETURN n * Fakultaet(n-1); (* n = 3 *)
❷  (* 1. rekursiver Aufruf Fakultaet(❸) *)
   IF n <= 1 THEN (* n = 2 *)
   ELSE RETURN n * Fakultaet(n-1); (* n = 2 *)
❶   (* 2. rekursiver Aufruf Fakultaet(❶) *)
    IF n <= 1 THEN (* n = 1 *)
    RETURN 1;
❷  (* zurück zur 1. rekursiven Prozedur Fakultaet(❷) *)
   RETURN 2 * 1;
❸ (* zurück zur 2. rekursiven Prozedur Fakultaet(❸) *)
  RETURN 3 * 2;
```

am Schluss: `a := 6;`

Besprechung Übungsserie 6 Aufgabe 1

„Alle Permutationen ausgeben“

```
PROCEDURE Step(elem : ARRAY OF INTEGER;
               level : INTEGER);
VAR i: INTEGER;
BEGIN
 IF level = nofElems-1 THEN
  FOR i := 0 TO nofElems-1 DO
   Out.Int(elem[i], 3)
  END;
  Out.Ln
 ELSE
  Step(elem, level+1);
  FOR i := level+1 TO nofElems-1 DO
   Swap(elem[level], elem[i]);
   Step(elem, level+1);
  END
 END
END Step;
```

Besprechung Übungsserie 6 Aufgabe 2 – Teil 1

„Hilbertkurven zeichnen“

```
MODULE Hilbert;
IMPORT  Turtle, In, Out;
CONST   RIGHT = 270;
        LEFT = 90;
VAR     topLevel : INTEGER;
        distance : INTEGER;

PROCEDURE ToggleDirection(VAR direction : INTEGER);
BEGIN
 direction := (direction + 180) MOD 360
END ToggleDirection;

PROCEDURE Draw*();
BEGIN
 Turtle.SetTo(distance, distance, 90);
 InnerDraw(topLevel, RIGHT)
END Draw;
```

Besprechung Übungsserie 6 Aufgabe 2 – Teil 2

„Hilbertkurve H = 1 zeichnen“

```
PROCEDURE InnerDraw(level : INTEGER;
                    direction : INTEGER);
BEGIN
 IF level = 1 THEN (* H = 1 *)
  Turtle.Walk(distance);
  Turtle.Turn(direction);
  Turtle.Walk(distance);
  Turtle.Turn(direction);
  Turtle.Walk(distance);
 ELSE

...
```

Besprechung Übungsserie 6 Aufgabe 2 – Teil 3

„Hilbertkurve H > 1 zeichnen“

```
 ELSE (* H > 1 *)
  Turtle.Turn(direction);
  ToggleDirection(direction);
  InnerDraw(level-1, direction);
  ToggleDirection(direction);
  Turtle.Turn(direction); Turtle.Walk(distance);
  InnerDraw(level-1, direction);
  ToggleDirection(direction);
  Turtle.Turn(direction);
  Turtle.Walk(distance); Turtle.Turn(direction);
  ToggleDirection(direction);
  InnerDraw(level-1, direction);
  Turtle.Walk(distance); Turtle.Turn(direction);
  ToggleDirection(direction);
  InnerDraw(level-1, direction);
  ToggleDirection(direction); Turtle.Turn(direction)
 END
END InnerDraw;
```