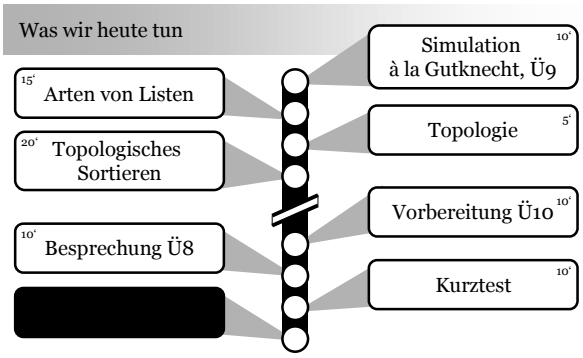
Informatik I WS 02/03

21. Januar 2003

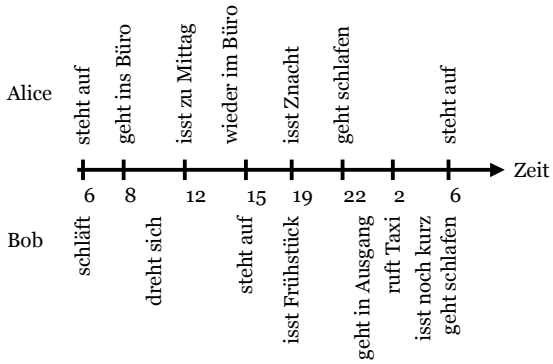
Heute volles Programm

Was wir heute tun



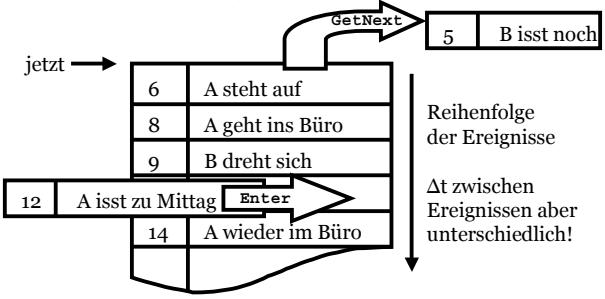
Simulation à la Gutknecht

in der Welt laufen Dinge asynchron ab!



Reihenfolge der Ereignisse

wir brauchen ein „Drehbuch des Schicksals“




```
PROCEDURE Enter( x : Object; t: REAL );
PROCEDURE GetNext( VAR x : Object );
```

Der Kreis schliesst sich

```
PutC( c: Client );   (* Kunde in Schlange setzen *)
GetC( VAR c: Client );   (* nächsten Kunde holen *)
PutS( c: Server );         (* freie Kasse melden *)
GetS( VAR c: Server );      (* freie Kasse holen *)
```

zwei Ereignisse: (1) Kunde kommt, (2) Kasse fertig

```
PROCEDURE Simulation*(); BEGIN
Enter( c, 0 );        (* Ersten Kunden erstellen *)
GetNext( x );                  (* Ersten holen *)
WHILE x.t < dauer DO         (* Zeit abgelaufen? *)
  now := x.t;                    (* Zeitsprung *)
  IF x IS Client THEN        (* was kommt jetzt? *)
    Arrive( x );            (* Ankunft behandeln *)
  ELSE
    End( x );           (* Kunde ist abgefertigt *)
  END;
  GetNext( x );             (* nächstes Ereignis *)
END;
END Simulation;
```

Handeln bei Ereignissen

was tun, wenn ein Kunde ankommt?

```
PROCEDURE Arrive( x: Client );
VAR s: Server; c: Client; tS, tC: REAL;
BEGIN
  x.arrT := now;          (* Kunde kommt jetzt an *)
  GetS( s );            (* schauen, obs Kasse hat *)
  IF s # NIL THEN               (* ist einer da? *)
    s.curS := x;            (* bediene den Kunden *)
    tS := RandomNumbers.Exp(2);   (* = 2*e^(-2*t) *)
    Enter( s, now + tS); (* wird in . fertig sein *)
  ELSE                        (* keine Kasse frei *)
    PutC( x );                (* in Warteschlange *)
  END;
  (* damit Kunden nicht aussterben: *)
  NEW( c );                 (* erstell neuen Kunden *)
  tS := RandomNumbers.Exp(2);
  Enter( c, now + tC ); (* wird irgendwann kommen *)
END Arrive;
```

Handeln bei Ereignissen

was tun, wenn eine Kasse einen Kunden abgehandelt hat

```
PROCEDURE End( x: Server );
VAR s: Server; c: Client;
BEGIN
  INC(nofC);      (* Statistik, wieviele bedient *)
  totTime := totTime + now - x.curC.arrT;
  GetC(c);               (* nächster Kunde holen *)
  IF c #NIL THEN            (* wenn einer da ist *)
    x.curC :=c;               (* bediene diesen *)
    Enter(x,now +tS)    (* wird in . fertig sein *)
  ELSE                        (* wenn keiner da ist *)
    PutS(x)           (* schick die Kasse warten *)
  END
END Arrive;
```

Arten von Listen

Listen waren bisher eine Anreihung von Elementen
gibt es aber noch Anderes?

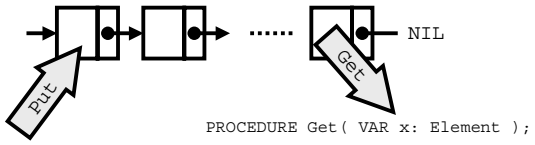
- Warteschlangen
- Ranglisten
- Stapel, etc.

auf diese Listen wollen wir andere Operationen ausführen

- hinten Einfügen, vorne Wegnehmen
- Einfügen nach Rangordnung
- was man oben drauf tut, bekommt man als Nächstes

Warteschlangen

keiner drängelt!



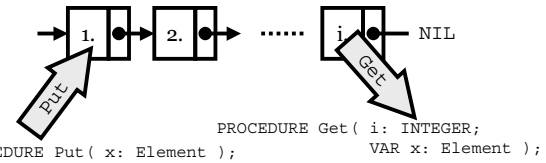

```
PROCEDURE Get( VAR x: Element );
PROCEDURE Put( x: Element );
```

was muss man sich intern merken?
reicht es, wenn man sich nur das letzte Element merkt?

*„**FIFO** – First In – First Out“ oder*
*„**LILO** – Last In – Last Out“*

Rangliste

jeder einordnen nach seinem Rang

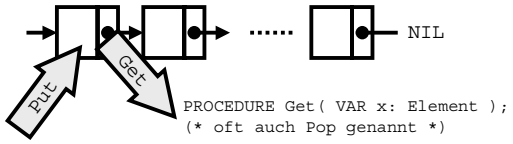



```
PROCEDURE Get( i: INTEGER;
               VAR x: Element );
PROCEDURE Put( x: Element );
```

wohin kommt ein Element x?
inwiefern ist die Warteschlange mit
der Rangliste zu vergleichen?

Stapel

was zuletzt draufgelegt wurde, kommt zuerst zurück




```
PROCEDURE Get( VAR x: Element );
(* oft auch Pop genannt *)
PROCEDURE Put( x: Element );
```

wie komme ich an das unterste Element?
schränken Stapel nicht zu sehr ein? Vorteile?
wann braucht man Stapel?

*„**LIFO** – Last In – First Out“* oder
*„**FILO** – First In – Last Out“*

Topologie

was bedeutet Topologie?

Topologie = Ordnung

Bsp. Rangliste

1. kommt vor 2. !
2. kommt vor 4. !
2. kommt vor 3. !
3. kommt vor 4., etc.

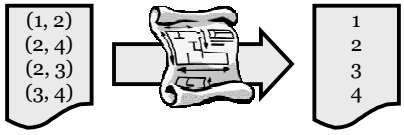
wie gelangt man von diesen Informationen / Regeln zur endgültigen Ordnung?

„A kommt vor B“ oder A ∠ B oder (A, B)

Topologisches Sortieren

gegeben: n Tupel in Form von (x, y) bzw. (x ∠ y)

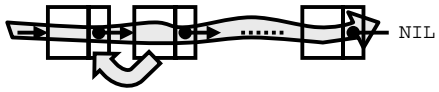
gesucht: lineare totale Ordnung < mit „a < b < ... < c“, so dass für alle Elemente a, b gilt: wenn a ∠ b, dann auch a < b



Datenstruktur der Eingabe | Interene Datenstruktur | Datenstruktur der Ausgabe / Externalisierung

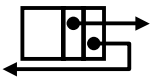
Einschub: Zusätzliche Pointer in Listen

Lineare Listen enthalten jeweils nur Pointer auf Nächsten
wie läuft man rückwärts in der Liste?

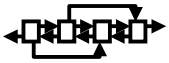


neuer Pointer nötig (*Doppelverkettete Liste*):

```
Element = POINTER TO RECORD
      wert : INTEGER;
      naechster : Element;
      vorheriger : Element; END;
```



nach Lust und Laune erweiterbar / modifizierbar:

 Übernächster, Überübernächster, etc.

Probleme beim Einfügen & Entfernen?

Topologisches Sortieren à la Gutknecht

Interne Datenstruktur

```
TYPE
Element = POINTER TO RECORD
      bezeichner : CHAR;
      kommtvor : Element; (* bezeichner kommt vor...
      zaehler : INTEGER;
      links : Link;
END;

Link = POINTER TO RECORD
      kommtvor : Element;
      naechsterlink : Link;
END;
```

Topologisches Sortieren –Internalisierung

```
PROCEDURE internalisieren();
VAR   pred, succ : CHAR; link : Link;
      predE, succE : Element;
BEGIN
WHILE noch mehr Tupel da DO
 lies nächsten Tupel ( pred, succ ) ein;
 IF pred existiert schon THEN predE := finde( pred );
 ELSE NEW( predE ); END;
 IF succ existiert schon THEN succE := finde( succ );
 ELSE NEW( succE ); END;
 predE.kommtvor := succE;
 NEW( link );
 neuerlink( predE, link ); link.kommtvor := succE;
 INC( succE.zaehler );
END;
END internalisieren;
```

Topologisches Sortieren - Externalisierung

```
PROCEDURE externalisierung();
VAR     min, cur: Element;
BEGIN
WHILE noch mehr Elemente da DO
 min := ErstesElementMitKleinstemZaehler();
 Entferne( min );
 Out.Char( min.bezeichner );
 IF min.zaehler # 0 THEN (* Fehler, weil Zyklus drinnen *)
 ELSE
   WHILE noch mehr verbundene Elemente da DO
    cur := naechstesVerbundenesE( min.kommtvor );
    DEC( cur.zaehler );
   END;
 END;
END;
END externalisierung;
```

Notizen

Vorbereitung Übungsserie 10

- dynamische Datenstruktur zur Darstellung der laufenden Rangliste während des Wettbewerbes.
- Implementieren Sie Prozeduren "Update" und "Print", welche die Position eines Teilnehmers in der Rangliste nach der Absolvierung einer Etappe in Abhängigkeit der erhaltenen Strafpunkte nachführt bzw. welche die laufende Rangliste ausdruckt.
- Benutzen Sie einen Zufallszahlengenerator zur Bestimmung der Anzahl Strafpunkte jedes Teilnehmers in jeder Etappe.

wie wird Rangliste implementiert?
wie fügt man in die Rangliste ein?
was passiert in der Prozedur Update?

Besprechung Übungsserie 8 Aufgabe 1

Reduktion von Bag auf Set

Bag hat welche Eigenschaften? Und das Set?

```
Init();
Append(v : INTEGER);
```

Was passiert, wenn mehrmals das v eingefügt wird?

```
Delete(v : INTEGER);
```

Was passiert, wenn das v mehrmals existiert?

```
Exists(v : INTEGER) : BOOLEAN;
```

Besprechung Übungsserie 8 Aufgabe 2 – Teil 1

- Prozesse sitzen im Kreis und reden miteinander.
- bis der Nachbar seinen Nachbar hört, vergeht 1 Sekunde
- um nach Le Lann-Chang-Roberts zu entscheiden, braucht ein Prozess 0 Sekunden

```
TYPE Prozess = POINTER TO RECORD
 in : INTEGER;          (* eingetroffene Meldung *)
 eigene : INTEGER;      (* eigener Wert *)
 out : INTEGER;         (* ausgehende Meldung *)
 naechster : Prozess;
END;
PROCEDURE Starten()*
VAR i : INTEGER; fertig : BOOLEAN;
BEGIN
 REPEAT
  fertig := Entscheiden();
  Sprechen();
 UNTIL fertig;
END Starten;
```

Besprechung Übungsserie 8 Aufgabe 2 – Teil 2

```
PROCEDURE Entscheiden() : BOOLEAN;
VAR akt : Prozess;
BEGIN
 akt := ringliste;
 REPEAT
  IF Entscheide( akt ) THEN RETURN TRUE;
  akt := akt.naechster;
 UNTIL akt = ringliste;
 RETURN FALSE;
END;
PROCEDURE Entscheide(p : Prozess) : BOOLEAN;
BEGIN
 IF    p.in < p.eigene THEN   RETURN FALSE;
 ELSIF p.in = p.eigene THEN
  Out.Integer( p.eigene, 2 ); RETURN TRUE;
 ELSIF p.in > p.eigene THEN
  p.out := p.in;              RETURN FALSE;
 END;
END Entscheide;
```

Besprechung Übungsserie 8 Aufgabe 2 – Teil 3

```
PROCEDURE Sprechen();
VAR akt, naechster : Prozess;
BEGIN
 akt := ringliste;
 REPEAT
  naechster := akt.naechster;
  naechster.in := akt.out;
  akt := akt.naechster;
 UNTIL akt = ringliste;
END;
```