Uebungsstunde vom 11.1.02, Info I

10. Januar 2002

1 Dynamische Datenstrukturen

Gründe für die Benutzung von dynamischen Datenstrukturen:

-
-
-

Typdefinition:

```
TYPE  Elem=POINTER TO Elemdesc;
      Elemdesc=RECORD
              data: INTEGER;
              next: Elem;
              END;

      List=RECORD
              first, last: Elem;
              END;
```

Zahl einlesen und in Elem-Kästchen einfügen:

```
VAR     eingabe:INTEGER; (* Hier wird im Speicher Platz für Zahl bereitgestellt*)
        elem:Elem;       (* Hier wird nur Platz für einen Pfeil auf ein Elem
                            bereitgestellt, nicht für den ganzen RECORD! *)




BEGIN
        In.Int(eingabe);
        NEW(elem);       (* Jetzt wird Speicherplatz für RECORD alloziert *)
        elem^.data:=eingabe;
        elem^.next:=NIL;
```


Beispiel für lineare Liste:

```
MODULE LinListDemo;

        IMPORT In, Out;
        TYPE    String=ARRAY 32 OF CHAR;

                Elem=POINTER TO Elemdesc;
                Elemdesc=RECORD
                        Wort: String;
                        next: Elem;
                        END;

                List=RECORD
                        first, last: Elem;
                        END;



        PROCEDURE InsertFront(VAR Liste:List; word:String);
                VAR data:Elem;
                BEGIN
                        NEW(data);
                        data^.Wort:=word;
                        IF (Liste.first=NIL) THEN Liste.last:=data; END;
                        data^.next:=Liste.first;
                        Liste.first:=data;
                END InsertFront;



        PROCEDURE Print(Liste:List);
                VAR temp: Elem;
                BEGIN
                        temp:=Liste.first;
                        WHILE (temp#NIL) DO
                                Out.String(temp^.Wort); Out.String("  ");
                                temp:=temp^.next;
                        END;
                        Out.Ln;
                END Print;



        PROCEDURE DeleteFront(VAR Liste:List);
                BEGIN
                        IF (Liste.first#NIL) THEN
                                IF (Liste.first=Liste.last) THEN Liste.last:=NIL; END;
                                Liste.first:=Liste.first^.next;
                        END;
                END DeleteFront;
```

```
PROCEDURE InsertEnd(VAR Liste:List; word: String);
        VAR data:Elem;
        BEGIN








        END InsertEnd;


 PROCEDURE Count(Liste:List):INTEGER;
        VAR temp: Elem; NofElem: INTEGER;
        BEGIN










        END Count;

 PROCEDURE DeleteEnd(VAR Liste:List);
        VAR temp, prev: Elem;
        BEGIN










        END DeleteEnd;
```



```
        PROCEDURE Test*;
                VAR word: String;
                                Liste:List;
                BEGIN
                        In.Open;
                        In.String(word);
                        WHILE In.Done DO
                                InsertEnd(Liste, word);
                                In.String(word);
                        END;
                        Out.Int(Count(Liste),8); Out.Ln;
                        Print(Liste);
                        DeleteFront(Liste);
                        Out.Int(Count(Liste),8); Out.Ln;
                        Print(Liste);
                        DeleteEnd(Liste);
                        Out.Int(Count(Liste),8); Out.Ln;
                        Print(Liste);
                END Test;

END LinListDemo.

LinListDemo.Test "So" "sollte" "das" "funktionieren"~
System.Free LinListDemo~
Output:
            4
            So  sollte  das  funktionieren
            3
            sollte  das  funktionieren
            2
            sollte  das
```

2 Bewertungskriterien Uebung 6

2.1 Permutation

- Rekursion
- Uebergabe der bisherigen Elemente by Value
- Schleife, die in jedem Step alle verbleibenden Rekursionen aufruft
- Abbruch, wenn Länge erreicht
- Uebergabe der aktuellen Position

2.2 Hilbert

- Rekursion
- Uebergabe der Rekursionsstufe
- Abbruchbedingung wenn Stufe 1 -> Zeichnen
- Wenn Stufe >1: Aufruf der unteren Stufe

2.3 Prisoner

- passende sinnvolle Datenstruktur (2/3 dim Array, Set)
- Rekursion
- Datenstruktur als Var Parameter übergeben oder global
- Uebergabe Tag, Position als value Parameter
- bei Sackgasse zurücksetzen des Versuchs

2.4 Springerkreis

- Rekursion
- Boolsche Rückgabe
- Uebergabe der Koordinaten
- Bei Nichtgelingen des Zugs wieder zurücksetzen
- Abbruchbedingung falls Kreis gefunden
- Abbruchbedingung falls ausserhalb Feld vor Zugriff
- Abbruchbedingung falls Feld schon besetzt

3 Uebung 8

3.1 Zusätzliche Aufgabe als Einstieg in die Programmierung mit dynamischen Datenstrukturen

In einer linearen Liste sollen die Namen von Personen (als String) mit ihrer jeweiligen Koerpergroesse in cm (als INTEGER) eingefuegt werden koennen. Die Liste soll nach Groesse geordnet sein, d.h. der Eintrag bei Liste.first ist die kleinste Person.

Diese Funktionen sollen implementiert werden:

- Einfuegen am richtigen Ort der Groesse entsprechend

```
PROCEDURE insert(VAR liste:Liste; name:String;
                     groesse; INTEGER);
```

- Ausgabe der ganzen Liste

```
PROCEDURE print(liste: Liste);
```

- Loeschen eines bestimmten Eintrages. Der Name der Person wird als Parameter gegeben

```
PROCEDURE delete(VAR liste:Liste; name:String);
```

- Zaehlen der Anzahl Eintraege in der Liste

```
PROCEDURE count(liste:Liste):INTEGER;
```

- Loeschen aller Eintraege von Personen, die groesser als eine bestimmte Groesse sind

```
PROCEDURE taller(VAR liste:Liste; groesse:INTEGER);
```

- Loeschen aller Eintraege von Personen, die kleiner als eine bestimmte Groesse sind

```
PROCEDURE smaller(VAR liste:Liste; groesse:INTEGER);
```

3.2 Client-Server-Simulation

Datenstruktur: 3 lineare Listen

Inhalt	Typ	wo einhängen?	wo rausnehmen?
Client-Warteschlange			
Free-Server-Liste			
Event-Kalender			

Datendefinitionen:

```
TYPE
  Object = POINTER TO ObjectRec;
  ObjectRec = RECORD
      next: Object;
      t: REAL (*activation time*)
  END;
```

```
Client = POINTER TO ClientRec;
ClientRec = RECORD (ObjectRec)
    arrT: REAL (*arrival time*)
END;

Server = POINTER TO ServerRec;
ServerRec = RECORD (ObjectRec)
    lineNo: INTEGER; (*number of served client waiting line*)
    curC: Client (*current client*)
END;
```

Abarbeiten des Event-Kalenders:

Typecast Wie kann ich ihm versichern, dass ein Objekt tatsächlich ein Client ist?

```
VAR     c:Client; o:Object;
BEGIN
        NEW(c);
        o:=c;
        o(Client)^.arrT:=14.5;
```