(* **Service-Modul ADTListe**

Version der ThiLLM-CD, überarbeitet von Michael Kühn (www.mksoftware.de.vu). Es gibt nun einen geändertem Inhaltstyp, sodass nicht nur INTEGER-Werte möglich sind, sondern beliebige Typen. Voreingestellt ist `ARRAY 10 OF CHAR`. Eine Änderung dieses Datentyps muss direkt im Modul ADTListe (Datei adtliste.mod) erfolgen.

Das Service-Modul muss eingebunden werden mit
`IMPORT ..., ADTListe;` *)

```
MODULE  ADTListe;
TYPE    TInhalt* = ARRAY 10 OF CHAR;
            (* Dieser Typ kann hier geändert werden! *)
        TListe*    = RECORD
                     END;
             (* Dies ist ein RECORD-Typ mit drei Zeigern, die vor dem Anwendungsprogrammierer versteckt
             werden. Es gibt je einen Zeiger zum ersten, zum letzten und zum aktuellen Listenelement.*)
PROCEDURE Erzeugen*(VAR liste: TListe);
             (* Es wird eine Liste erzeugt. Dies ist bei dynamischen Datenstrukturen immer nötig. Eine Definition
             mit VAR meineListe: ADTListe.TListe; genügt nicht. Es muss mit
             Erzeugen(meineListe) noch eine Instanz des abstrakten Datentyps kreiert werden. *)
PROCEDURE ListeLeer*(liste: TListe): BOOLEAN;
             (* Es wird geprüft, ob die Liste leer ist. *)
PROCEDURE ListenEnde*(liste: TListe): BOOLEAN;
             (* Diese Funktion liefert den Wert TRUE, wenn das aktuelle Element das letzte der Liste ist oder wenn
             die Liste leer ist. Ansonsten ist der Wert FALSE. *)
PROCEDURE EinfuegenElement*(VAR liste: TListe; inhalt: TInhalt);
             (* Es wird in die liste ein neues Element mit dem angegebenen inhalt eingefügt. Dabei wird das
             neue Element vor das gerade aktuelle Element eingefügt. Wenn die Liste zuvor leer war oder nur aus
             einem Element bestand, wird das neue Element zum ersten der Liste. Nach dieser Operation ist das neu
             eingefügte Element auch das aktuelle. *)
PROCEDURE AnhaengenElement*(VAR liste: TListe; inhalt: TInhalt);
             (* Es wird am Ende der liste ein neues Element mit dem angegebenen inhalt eingefügt. Sollte die
             Liste leer sein, wird das neue Element zum ersten. Das neue Listen-Element wird dabei auch zum
             aktuellen Element. *)
PROCEDURE LoeschenElement*(VAR liste: TListe);
             (* Das aktuelle Element der liste wird gelöscht, die Liste ist anschließend um ein Element verkürzt.
             Das aktuelle Element der Liste ist nach dieser Operation das erste, sofern vorhanden. *)
PROCEDURE GeheErstes*(VAR liste: TListe);
             (* Verschiebt den Zeiger des aktuellen Elements auf das erste der liste. Sollte die Liste leer sein,
             passiert nichts (auch keine Fehlermeldung oder Programmabbruch). *)
PROCEDURE GeheLetztes*(VAR liste: TListe);
             (* Verschiebt den Zeiger des aktuellen Elements auf das letzte der liste. Sollte die Liste leer sein,
             passiert nichts (auch keine Fehlermeldung oder Programmabbruch). *)
PROCEDURE GeheNaechstes*(VAR liste: TListe);
             (* Verschiebt den Zeiger des aktuellen Elements eine Position in der liste weiter. Wenn das aktuelle
             Element schon das letzte war, wird eine Fehlermeldung ausgegeben und das Programm abgebrochen.
             Der Anwendungsprogrammierer muss dafür sorgen, dass dieser Fall nicht eintritt! *)
PROCEDURE HoleEintrag*(liste: TListe; VAR inhalt: TInhalt);
             (* Die Daten des aktuellen Elements werden in der Variablen inhalt zurückgegeben. Sollte die Liste
             leer sein, gibt es eine Fehlermeldung und das Programm wird abgebrochen. Der Anwendungsprogram-
             mierer muss dafür sorgen, dass dieser Fall nicht eintritt! *)
PROCEDURE SchreibeEintrag*(liste: TListe; inhalt: TInhalt);
             (* Die Daten des aktuellen Elementes werden mit denen im Parameter inhalt übergegebenen Daten
             überschrieben. Sollte die Liste leer sein, gibt es eine Fehlermeldung und das Programm wird abge-
             brochen. Der Anwendungsprogrammierer muss dafür sorgen, dass dieser Fall nicht eintritt! *)
END ADTListe.
```