Bibliothek in Modula-2
(Typen, Klassen, Prozeduren)

© Beat Heil
Chasernweg 20, CH-8302 Kloten

INHALT

Vorwort

Im Jahre 1990 suchte ich nach Literatur über Software-Entwicklungs-Umgebungen. Dabei bin ich auf einen Bericht über das «IPSEN-Projekt (Incremental Programming Support Environment)» an der Rheinisch-Westfälischen Technischen Hochschule Aachen gestossen [LEW], in welchem auf Seite 29 festgestellt wird: «Bei der Verwendung von Modula-2 als Implementierungssprache gelingt die Transformation der Systemarchitektur in eine Implementierungsvorgabe wesentlich besser als in C». Die Systemarchitektur von IPSEN orientiert sich am Definitions-Modul von Modula-2, was eine klare Trennung zwischen Design (Programmieren im Grossen) und Implementierung (Programmieren im Kleinen) bedeutet. Ich beschloss deshalb, mich mit dieser Sprache im Detail zu befassen.

Nun ist Theorie jeweils noch keine Praxis, obwohl alleine schon das Konzept von Modula-2 einen hervorragenden Eindruck hinterliess. Ich wollte wissen, wie es sich in der Anwendung verhält. Dazu besorgte ich mir zwei Bücher [WIR],[SAL] und TopSpeed Modula-2 (OO) als Entwicklungsystem.

Mein Ziel war, ein vollständiges, modernes System inkrementell zu entwerfen und zu implementieren um die Konzepte in der Praxis zu erproben, und um teilweise eigene, neue Lösungsansätze entwickeln zu können. Als Abfallprodukt ist so umfangreiche Software entstanden, welche meinem persönlichen Zweck dient. Dank der Basis-Module konnten sie in relativ kurzer Zeit realisiert werden. Das inkrementelle Vorgehen liess sich mit Modula-2 ausgezeichnet realisieren, sei es konventionell oder objektorientiert. Dabei galt es jeweils Vor- und Nachteile der beiden Design-Methoden sorgfältig gegeneinander abzuwägen. Die Software liess sich problemlos an neue Technologien anpassen (z.B. Einbau der Mausereignisse). Heute läuft sie im DOS-Fenster unter Windows98. Das vorliegende Dokument behandelt diese Basis-Module und ausgewählte Beispiele.
Zusammenfassend darf (auch) ich feststellen: das Konzept von Modula-2 ist genial einfach - die Umkehrung gilt natürlich ebenso. Es hat mir ausserordentlich viel Freude bereitet. Dafür danke ich dem Vater von Modula-2, Niklaus Wirth, ganz herzlich und gratuliere Ihm zu seiner fundierten Arbeit, welche sich im neuesten Kind „Oberon“ fortsetzt.

2. August 1996 Beat Heil

[LEW] Lewerentz Claus, Lehrstuhl für Informatik III, RWTH Aachen. Interaktives Entwerfen grosser Programmsysteme, Informatik-Fachberichte 194, Springer-Verlag, ISBN 3-540-50553-9

[SAL] Sale Arthur, Prof. Hobart, Tasmanien. Modula2 - Durch systematischen Entwurf zum korrekten Programm, ISBN 3-925118-58-6.

[WIR] Wirth Niklaus, Prof. Dr. Dr. h.c., ETH-Zürich. Algorithmen und Datenstrukturen mit Modula-2, ISBN 3-519-02260-5.

Com_1

Einleitung

Dieser Modul stellt allgemein gehaltene Funktionen zur Verfügung, wie sie in den meisten Anwendungen benötigt werden.

Beschreibung

```
FROM Str IMPORT CHARSET;

CONST
        TextSet = CHARSET{'A'..'Z','a'..'z','0'..'9',
                          'Ä','ä','à','â','á','å','Å',
                                'ç','Ç',
                                'é','è','ê','ë',
                                'ï','î','ì','í',
                                'Ö','ö','ô','ò','ó',
                                'Ü','ü','û','ù','ú',
                                'ÿ'};
TYPE
        tLine = ARRAY[0..255] OF CHAR;
        tLongReal = ARRAY[0..23] OF CHAR;
        tName = ARRAY[0..11] OF CHAR;
        tPath = ARRAY[0..77] OF CHAR;
        tPict = (Left,Right, Zero(*suppress*), Point,Comma,Blank,Quote);
        rBM = RECORD
           PN : CARDINAL;
           PD : ARRAY[0..255] OF CARDINAL;
        END;
        sChar = SET OF CHAR;
        sPict = SET OF tPict;
CONST
        EOL = sChar{CHR(0),CHR(10),CHR(13)};

        MaxC  = LONGREAL(MAX(CARDINAL));
        MaxLC = LONGREAL(MAX(LONGCARD));
        MaxSC = LONGREAL(MAX(SHORTCARD));
        MinI  = LONGREAL(MAX(INTEGER)) * (-1.0);
        MaxI  = LONGREAL(MAX(INTEGER));
        MinLI = LONGREAL(MAX(LONGINT)) * (-1.0);
        MaxLI = LONGREAL(MAX(LONGINT));
        MinSI = LONGREAL(MAX(SHORTINT)) * (-1.0);
        MaxSI = LONGREAL(MAX(SHORTINT));
        MinR  = LONGREAL(MAX(REAL)) * (-1.0);
        MaxR  = LONGREAL(MAX(REAL));
        MinLR = MAX(LONGREAL) * (-1.0);
        MaxLR = MAX(LONGREAL);

        PicLP   = sPict{Left,Point};                (* -zzz9.99  *)
        PicLPB  = sPict{Left,Point,Blank};          (* -z zz9.99 *)
        PicLPQ  = sPict{Left,Point,Quote};          (* -z'zz9.99 *)
        PicLPZ  = sPict{Left,Point,Zero};           (* -zzzZ.ZZ  *)
        PicLPZB = sPict{Left,Point,Zero,Blank};     (* -z zzZ.ZZ *)
        PicLPZQ = sPict{Left,Point,Zero,Quote};     (* -z'zzZ.ZZ *)
        PicRP   = sPict{Right,Point};               (* zzz9.99-  *)
        PicRPB  = sPict{Right,Point,Blank};         (* z zz9.99- *)
        PicRPQ  = sPict{Right,Point,Quote};         (* z'zz9.99- *)
        PicRPZ  = sPict{Right,Point,Zero};          (* zzzZ.ZZ-  *)
        PicRPZB = sPict{Right,Point,Zero,Blank};    (* z zzZ.ZZ- *)
        PicRPZQ = sPict{Right,Point,Zero,Quote};    (* z'zzZ.ZZ- *)
```

PROCEDURE ChrCap(Chr:CHAR):CHAR;
Wandelt Klein- in Grossbuchstaben um. So wird z.B. «ü» zu «Ü».

PROCEDURE ChrsCap(VAR Chrs:ARRAY OF CHAR);
Wandelt Klein- in Grossbuchstaben um. So wird z.B. «müller» zu «MÜLLER».

PROCEDURE ChrsKey(VAR Chrs:ARRAY OF CHAR);
Wandelt Klein- in Grossbuchstaben um. Umlaute werden dabei aufgeschlüsselt (Keyformat). So wird z.B. «müller» zu «MUELLER».

PROCEDURE ChrC(Chr:ARRAY OF CHAR;From,To:CARDINAL):CARDINAL;
Exrahiert CARDINAL aus «Chr» . «From» und «To» mit 0..MAX(CARDINAL)-1 begrenzen den Bereich.
Z.B: r=1255 => r:=ChrC('Wert 1'255.90- pro Einheit',0,MAX(CARDINAL));

PROCEDURE ChrChr(S:ARRAY OF CHAR; From:CARDINAL;
T:ARRAY OF CHAR; VAR R:ARRAY OF CHAR);
Extrahiert «Result» aus «Source», beginnend mit «From» und endend bei einem der «Token« , bzw. am Ende von «Source». Für «From» gilt 0..MAX(CARDINAL) -1.

PROCEDURE ChrExcl(S:ARRAY OF CHAR; From:CARDINAL;
Begin,End:ARRAY OF CHAR;
VAR R:ARRAY OF CHAR):CARDINAL;
Extrahiert «Result» aus «Source», und zwar jener Bereich, welcher zwischen «Begin» und «End» liegt. Der Vergleich basiert auf Grossbuchstaben (ChrsCap). Für «From» gilt 0..MAX(CARDINAL)-1. Die aktuelle Position von «Begin» wird als Resultat zurückgegeben, bzw. MAX(CARDINAL), wenn nicht extrahiert wurde.

PROCEDURE ChrIncl(S:ARRAY OF CHAR; From:CARDINAL;
Begin,End:ARRAY OF CHAR;
VAR R:ARRAY OF CHAR):CARDINAL;
Extrahiert «Result» aus «Source», und zwar jener Bereich, welcher «Begin» und «End» einschliesst. Der Vergleich basiert auf Grossbuchstaben (ChrsCap). Für «From» gilt 0..MAX(CARDINAL)-1. Die aktuelle Position von «Begin» wird als Resultat zurückgegeben, bzw. MAX(CARDINAL), wenn nicht extrahiert wurde.

PROCEDURE ChrLC(Chr:ARRAY OF CHAR;From,To:CARDINAL):LONGCARD;
Extrahiert LONGCARDINAL aus «Chr». «From» und «To» mit 0..MAX(CARDINAL) begrenzen den Bereich.
Z.B: r=7246255 => r:=ChrLC('Wert7'246'255.90- pro Einheit',0,MAX(CARDINAL));

PROCEDURE ChrLI(Chr:ARRAY OF CHAR;From,To:CARDINAL):LONGINT;
Extrahiert LONGINTEGER aus «Chr» . «From» und «To» mit 0..MAX(CARDINAL) begrenzen den Bereich.
Z.B: r=-7246255 => r:=ChrLI('Wert7'246'255.90- pro Einheit',0,MAX(CARDINAL));

PROCEDURE ChrLR(Chr:tLongReal; Body:REAL; Min,Max:LONGREAL;
VAR OK:BOOLEAN):LONGREAL;
Extrahiert LONGREAL aus «Chr» . Mit «Body» werden Vor-/Nachkommastellen und Vorzeichen angegeben. «Min» und «Max» geben die erlaubte Unter- und Obergrenze des Resultats an. Das Resultat entspricht allen verlangten Kriterien, wenn «OK» Wahr ist. Andernfalls ist das Resultat entsprechend angepasst.
Z.B: r=-46255.90 & ok=TRUE => r:=ChrLR('46'255.90-',-5.2,-50000,50000,ok);

PROCEDURE CloseChr(VAR Chr:ARRAY OF CHAR);
Ersetzt rechte Leerstellen, also CHR(32), durch CHR(0).

PROCEDURE CompareLx(New,Old:ARRAY OF CHAR):INTEGER;
Vergleicht «Old» mit «New» lexikalisch, d.h: primär im Keyformat (ChsKey) und sekundär im Originalformat. Als Ergebnis wird -1, 0, +1 (für kleiner, gleich, grösser) zurückgegeben. Z.B: Actual,actual,...

PROCEDURE CompileBM(VAR P:ARRAY OF CHAR):rBM;
Kompiliert «Pattern» für die schnelle Suche nach der Boyer-Moore-Methode (FindBM, FindBMc).

PROCEDURE Compress(VAR Line:ARRAY OF CHAR);
Komprimiert Textdaten durch zusammenfassen sich wiederholender Zeichen. Als Flag dient CHR(26).

PROCEDURE Compress8(VAR Line:ARRAY OF CHAR);
Komprimiert Textdaten durch setzen des Tabulatorflags CHR(9) für Leerstellen (i MOD 8).

PROCEDURE CountText(Chr:ARRAY OF CHAR):CARDINAL;
Zählt die belegten Stellen, ohne die rechts stehenden CHR(32). Z.B n=5 => n=CountText('xx xx ');

PROCEDURE CreateRoman(Nr:CARDINAL; Cap:BOOLEAN;
VAR Result:ARRAY OF CHAR):BOOLEAN;
Gibt in «Result» die römischen Ziffern von «Nr» aus, gemäss «Cap» =FALSE/TRUE in Klein-/Grossbuchstaben.

PROCEDURE Expand(VAR Line:ARRAY OF CHAR);
Expandiert mit Compress behandelte Textdaten.

PROCEDURE Expand8(VAR Line:ARRAY OF CHAR);
Expandiert mit Compress8 behandelte Textdaten.

PROCEDURE FindBM(VAR S,P:ARRAY OF CHAR; s:CARDINAL):CARDINAL;
Ermittelt Position 0..MAX(CARDINAL)-1 von «Pattern» in «Source», wobei mit der Suche in Position «s» gestartet wird. Wurde das Muster nicht gefunden, wird MAX(CARDINAL) zurückgegeben. Für die Suche wird ein Algorithmus nach Boyer-Moore verwendet. Falls das selbe Muster mehrmals gesucht wird, ist FindBMc geeigneter.

PROCEDURE FindBMc(VAR S,P:ARRAY OF CHAR; VAR BM:rBM;
s:CARDINAL): CARDINAL;
Entspricht FindBM mit dem Unterschied, dass «Pattern» mit CompileBM vorkompiliert, und in «BM» übergeben wird. Für eine einmalige Suche ist FindBM geeigneter.

PROCEDURE ItemS(VAR R:ARRAY OF CHAR; S,T:ARRAY OF CHAR;
N:CARDINAL):BOOLEAN;
Gibt in «Result» das «N-te» Element von «Source», aufgespalten durch eines der «Token», zurück. Für «N» gilt 0..MAX(CARDINAL)-1 und, im Unterschied zu Str.ItemS von Modula-2, zählen auch leere Elemente.
Z.B: r:='c' => ok:=ItemS('a/b//c',r,'/',3);

PROCEDURE LRChr(LR:LONGREAL; Body:REAL; Pict:sPict; To:CARDINAL;
VAR Chr:ARRAY OF CHAR; VAR OK:BOOLEAN);
Überträgt den Wert von «LongReal» nach «Chr» an die Position «To» . Das Ergebnis ist abhängig von «Body» für Vor-/Nachkommastellen und Vorzeichen, und von «Picture» für die Darstellung. In «OK» wird die Aktion bestätigt.
Z.B: Chr:='Ergebnis 1'235.45- pro Einheit' => LRChr(1235.45-,-4.2,sPict{Right,Point,Quote},9,Chr,ok);

PROCEDURE OpenChr(VAR Chr:ARRAY OF CHAR);
Füllt nach dem Textende mit CHR(32) auf.

PROCEDURE ShiftLeft(VAR Chr:ARRAY OF CHAR; By:CARDINAL);
Verschiebt den Inhalt von «Chr» um «By»-Stellen nach links.

PROCEDURE ShiftRight(VAR Chr:ARRAY OF CHAR; By:CARDINAL);
Verschiebt den Inhalt von «Chr» um «By»-Stellen nach rechts.

PROCEDURE SizeLR(Body:REAL; Pict:sPict; VAR DP:CARDINAL):CARDINAL;
Ermittelt die Anzahl Stellen welche benötigt werden, um einen Wert mittels «Body» und «Picture» darzustellen. In «DP» wird zudem die Position des Dezimalpunkts übermittelt (LRChr). Im Funktionswert wird das Resultat zurückgegeben (0 = Fehler).

Com_2

Einleitung

Dieser Modul stellt Funktionen für anonyme Variablen vom Typ Byte, Char und String zur Verfügung. Sie finden dort Verwendung, wo von den Daten nur noch deren Adresse und Länge bekannt sind.

Beschreibung

PROCEDURE CloseStr(Str:ADDRESS; Size:CARDINAL);
Ersetzt an der Adresse «Str» mit Länge «Size» rechte Leerstellen, also CHR(32), durch CHR(0);

PROCEDURE CompareByte(Byte1,Byte2:ARRAY OF BYTE):INTEGER;
Vergleicht «Byte1» mit «Byte2». Als Ergebnis wird -1, 0, +1 (für kleiner, gleich, grösser) zurückgegeben.

PROCEDURE CompareStr(Str1,Str2:ADDRESS; Size:CARDINAL):INTEGER;
Vergleicht «Str1» mit «Str2» im Originalformat und auf der Länge «Size» . Als Ergebnis wird -1, 0, +1 (für kleiner, gleich, grösser) zurückgegeben.

PROCEDURE CountText(Str:ADDRESS; Size:CARDINAL):CARDINAL;
Zählt an der Adresse «Str» mit Länge «Size» die belegten Stellen, ohne die rechts stehenden CHR(32).

PROCEDURE DeleteChr(Str:ADDRESS; Size,Pos:CARDINAL);
Entfernt im «Str» mit Länge «Size» die Position «Pos».

PROCEDURE DeleteToEnd(Str:ADDRESS; Size,From:CARDINAL);
Löscht im «Str» mit Länge «Size» alle Stellen ab Position «From»

PROCEDURE InsertChr(Chr:CHAR; Str:ADDRESS; Size,To:CARDINAL);
Fügt im «Str» mit Länge «Size» den «Chr» an der Position «To» ein.

PROCEDURE MoveChr(Chr:CHAR; Str:ADDRESS; Size,To:CARDINAL);
Überträgt im «Str» mit Länge «Size» den «Chr» in diePosition «To» .

PROCEDURE OpenStr(Str:ADDRESS; Size:CARDINAL);
Füllt nach dem Textende im «Str» mit Länge «Size» diesen mit CHR(32) auf.

CRf_1

Einleitung

Dieser Modul realisiert Funktionen für die Erstellung von Kreuzreferenzen und Wortlisten. Dazu kann eine beliebige Anzahl Filter eingesetzt werden, mit welchen sich Worte sowohl aus-, als auch einfiltern lassen.

Beschreibung

```
FROM Com_1 IMPORT tPath;
FROM Scr_1 IMPORT prPal;
FROM Tex_1 IMPORT pcText;

TYPE
        prFilter;

        CLASS cFilter;
           FD : prFilter;
           PROCEDURE Dispose;
           PROCEDURE Find(Word:ARRAY OF CHAR):BOOLEAN;
           PROCEDURE New;
           PROCEDURE Load(FP:tPath; Sign:CHAR;Format,Accept,Delete:ARRAY OF CHAR):BOOLEAN;
        END cFilter;
```

Erstellt und Verwaltet Filterdaten.

```
        PROCEDURE Dispose;
```

Entfernt alle Daten der Klasse und gibt den Speicher frei.

```
        PROCEDURE Find(Word:ARRAY OF CHAR):BOOLEAN;
```

Sucht «Word» im Filter (gemäss Formatangabe bei Load) und liefert im Funktionswert das Ergebnis.

```
        PROCEDURE New;
```

Initialisiert den Speicher für die Daten der Klasse.

```
        PROCEDURE Load(FP:tPath; Sign:CHAR;Format,Accept,Delete:ARRAY OF CHAR):BOOLEAN;
```

Erstellt mit der Datei gemäss «FP» den Filter und bestätigt dies im Funktionsergebnis. «Sign» bestimmt die Filterrichtung und «Format» den Schlüsselvergleich. In «Accept» können Sonderzeichen mitgegeben werden, die zur Wortbindung gehören (z.B. '_' für ein_wort) und in «Delete» solche, die vor der Wortbildung aus dem Text zu entfernen sind (z.B. '{}' für Aus{b}lenden). Für die Wortbildung ist Com_1.Textset massgebend. Trennzeichen am Zeilenende und in Tabellenrastern werden automatisch behandelt.
Sign: - = ausfiltern; + = einfiltern; Format: K = Key (Com_1.ChsKey); U = Upper (Com_1.ChsCap); S = Standard (unverändert)

```
TYPE
        prReference;

        CLASS cReference;
           RD : prReference;
           PROCEDURE CloseFile;
           PROCEDURE Dispose;
           PROCEDURE Empty():BOOLEAN;
           PROCEDURE FormRef(Mask:ARRAY OF CHAR; Text,Line:CARDINAL):BOOLEAN;
           PROCEDURE GetFile():tPath;
           PROCEDURE MakeRef(Line:ARRAY OF CHAR; Nr:LONGCARD; Filter:cFilter);
           PROCEDURE New(Tex:pcText; Pal:prPal);
           PROCEDURE OpenFile(FP:ARRAY OF CHAR);
        END cReference;
```

Erstellt wahlweise Wortlisten oder Kreuzreferenzen auf Seitennummern, Dateipositionen etc. Das Ergebnis kann (a) in eine Datei abgelegt, oder (b) direkt weiterverarbeitet werden (z.B. für Druckausgabe).

Ablauf (a): (cFilter), New, OpenFile, MakeRef, CloseFile, Dispose
Ablauf (b): (cFilter), New, MakeRef, FormRef(Loop), GetFile, Dispose

PROCEDURE CloseFile;
Schliesst die Datei, welche mit OpenFile eröffnet wurde.

PROCEDURE Dispose;
Entfernt alle Daten der Klasse und gibt den Speicher frei.

PROCEDURE Empty():BOOLEAN;
Beantwortet die Frage nach leerer Referenztabelle.

PROCEDURE FormRef(Mask:ARRAY OF CHAR; Text,Line:CARDINAL):BOOLEAN;
Lässt die bis dato angefallenen Referenzen abarbeiten (formen). Im Funktionsresultat wird die Aktion bestätigt. «Mask» bestimmt die Einteilung der Zeilen. In «Text» wird die Zeilenzahl, welche pro Seite für den Text (also ohne Kopf-/Fusszeilen) reserviert ist, bestimmt und in «Line» die Anzahl Zeichen, welche auf einer Zeile Platz finden.

Mask: ^ Spaltenrand links; # Spaltenrand rechts;
z.B. 3-spaltig: '^ .....# ^ .....# ^ .....#'

PROCEDURE GetFile():tPath;
Gibt Pfad und Name der Datei zurück, in welcher die Kreuzreferenzen abgelegt sind (OpenFile).

PROCEDURE MakeRef(Line:ARRAY OF CHAR; Nr:LONGCARD; Filter:cFilter);
Bildet mit dem Inhalt von «Line» Referenzen auf «Nr». Ist sie MAX(LONGCARD), wird lediglich eine Wortliste gebildet. Ohne «Filter» ist NIL anzugeben. Filter sind differenziert einsetzbar. Für die Wortbildung ist Com_1.Textset massgebend. Trennzeichen am Zeilenende und in Tabellenrastern werden automatisch behandelt.

PROCEDURE New(Tex:pcText; Pal:prPal);
Initialisiert den Speicher für die Daten der Klasse. Für den Timer (Scr_1.cTimer) sind Pointer auf Text und Farbpalette erforderlich.

PROCEDURE OpenFile(FP:ARRAY OF CHAR);
Sollen Kreuzreferenzen oder Wortlisten in eine bestimmte Datei geschrieben werden, muss sie hier eröffnet, und mit CloseFile geschlossen werden. Ansonsten wird mit Temporärdateien gearbeitet, welche automatisch abgehandelt sind.

Dat_1

Einleitung

Dieser Modul stellt Funktionen für alle Operationen mit Datum zur Verfügung. Die Jahreszahl wird intern grundsätzlich 4-stellig behandelt. Extern kann zwischen 2- und 4-stelliger Darstellung gewählt werden.

Die Datumaufbereitung von extern nach intern, bzw. umgekehrt, erfolgt jeweils anhand einer Maske. Diese definiert das Datum in beliebiger Zusammensetzung und Kombination von DD=Tag; MM=Monat; YY=Jahr 2-stellig; YYYY=Jahr 4-stellig; Als Interpunktionszeichen dienen Punkt, Binde- und Schrägstrich. Anstelle von «DMY» ist auch «dmy» zulässig. Z.B: 'DD.MM.YY' oder 'dd-mm-yyyy'.

Beschreibung

```
IMPORT Lib;

TYPE
        rDate = RECORD
           Year  : CARDINAL;
           Month : SHORTCARD;
           Day   : SHORTCARD;
         END;
         tDayOfWeek = Lib.DayType;
```

PROCEDURE CharDate(Char,Mask:ARRAY OF CHAR):rDate;
Extrahiert aus «Char» das Datum, wobei «Mask» das Ergebnis steuert (Einleitung), und liefert es im Funktionswert.

PROCEDURE Compare(Date1,Date2:rDate):INTEGER;
Vergleicht «Date1» mit «Date2» und gibt als Ergebnis -1, 0, +1 (für kleiner , gleich, grösser) zurück.

PROCEDURE DateChar(Date:rDate; Mask:ARRAY OF CHAR;VAR Char:ARRAY OF CHAR);
Bereitet aus «Date» und «Mask» (Einleitung) das externe Datum auf und stellt es in «Char».

PROCEDURE DifferAge(Date1,Date2:rDate):rDate;
Errechnet die Differenz zwischen «Date1» und «Date2» und gibt das Ergebnis in Jahre, Monate und Tage zurück. In welchem Parameter welches Datum steht, spielt keine Rolle. Die Funktion testet selber, welches Datum grösser, und welches kleiner ist.

PROCEDURE DifferDays(Date1,Date2:rDate):LONGCARD;
Errechnet die Differenz zwischen «Date1» und «Date2» und gibt das Ergebnis in Tage zurück. In welchem Parameter welches Datum steht, spielt keine Rolle. Die Funktion testet selber, welches Datum grösser, und welches kleiner ist.

PROCEDURE GetDayOfWeek(Date:rDate; Week:ARRAY OF CHAR;
VAR Day:ARRAY OF CHAR);
Ermittelt den Wochentag von «Date», holt das entsprechende Item aus «Week» und stellt es in «Day» zur Verfügung. Wichtig ist, dass die Items 0..6 als Sonntag..Samstag verwendet werden.

Week: z.B. ' Sonntag,Montag,Dienstag,Mittwoch,Donnerstag,Freitag,Samstag' oder
' Son,Mon,Die,Mit,Don,Fre,Sam'

PROCEDURE GetSystem():rDate;
Liefert als Funktionsergebnis das Systemdatum.

PROCEDURE MakeCentury(VAR Year:CARDINAL):BOOLEAN;
Ermittelt für eine 2-stellige, die 4-stellige Jahreszahl derart, dass: «Year» <= (Systemjahr+5). Falls der Ausgangswert in «Year» > 99 war, wird kein Ergebnis ermittelt und der Funktionswert FALSE zurückgegeben.

PROCEDURE Modify(Date:rDate; Days:LONGINT):rDate;
Modifiziert «Date» um den Wert «Days» (positiv oder negativ) und gibt als Ergebnis das neue Datum zurück.

PROCEDURE SplitMask(VAR Mask,Year,Month,Day:ARRAY OF CHAR):BOOLEAN;
Splittet «Mask» (Einleitung) in die Bestandteile «Year», «Month» und «Day» auf.

PROCEDURE Verify(Date:rDate):BOOLEAN;
Prüft «Date» und gibt das Ergebnis als Funktionswert zuück.

PROCEDURE WhichDayOfWeek(Date:rDate):tDayOfWeek;
Gibt den Wochentag von «Date» als Ergebnis (SHORTCARD(tDayOfWeek) = 0..6) zurück.

DOS_1

Einleitung

Dieser Modul stellt Funktionen für DOS-Belange zur Verfügung. Systemspezifische Änderungen müssen damit nur an dieser Stelle erfolgen.

Beschreibung

```
CONST
        Tickers = 18;
```

PROCEDURE Command():BOOLEAN;
Verzweigt auf die Command-Ebene, aus der mit «Exit» zurückgekehrt wird. Im Funktionswert wird die Aktion bestätigt.

PROCEDURE FatalError(Error:ARRAY OF CHAR);
Schreibt vor dem Abbruch die Fehlerbedingung «Error» in eine Temporärdatei «ERR*.$$$» .

PROCEDURE GetTicker():LONGCARD;
Greift auf die Echtzeituhr des Systems zu und gibt als Ergebnis den Tickerwert zurück. Er wird berechnet aus ((Register.CX * (60 * 60 * Tickers)) + Register.DX) und dient der Steuerung des Timers (Scr_1.cTimer) und der Namensgebung für Temporärdateien.

Eve_1

Einleitung

Dieser Modul stellt Funktionen für Keyboard- und Mausereignisse zur Verfügung. Er initialisiert beim Start Maus-Prozeduren ohne Funktion.

Beschreibung

```
IMPORT BiosIO, MsMouse, DOS_1;
FROM MsMouse IMPORT MsData, MsMotion;
FROM Window IMPORT RelCoord, WinType;

CONST
        MaxButton = 3;
        Mouse = CHR(255);
TYPE
        rMouse = RECORD
           State : CARDINAL;
           Button : ARRAY[0..MaxButton-1]OF CARDINAL;
        END;

        tEvent = (Ascii,Extended,Click1,Click2);
        rEvent = RECORD;
           Kind : tEvent;
           Key : CHAR;
           x,y : RelCoord;
         END;

        tCursorProc = PROCEDURE(BOOLEAN);
VAR
        Cursor : tCursorProc;
```

Cursor(TRUE); schaltet ihn ein, Cursor(FALSE) aus.

PROCEDURE EventClear(VAR Event:rEvent);

Löscht alle hängigen Ereignisse, wobei «Esc» und «AltX» behandelt werden.

PROCEDURE EventGet(W:WinType; VAR Event:rEvent):BOOLEAN;

Schaut nach einem hängigen Ereignis, ohne darauf zu warten, und zeigt dies im Ergebnis an.

PROCEDURE EventWait(W:WinType; VAR Event:rEvent);

Wartet auf ein Ereignis, ohne es zu behandeln.

PROCEDURE EventWaitGet(W:WinType; VAR Event:rEvent);

Wartet, bis ein Ereignis eintritt, und behandelt es.

PROCEDURE InitMouse(State:CARDINAL):CARDINAL;

Initialisiert Mausprozeduren gemäss «State» und gibt den tatsächlichen Status im Ergebnis zurück.

State: 0=inaktiv; 1=aktiv

PROCEDURE MouseToKey(VAR Event:rEvent; x1,x2,y:RelCoord;From:tEvent; To:CHAR);

Wandelt eine Maus- in eine Keyboardaktion um. Dies ist dann der Fall, wenn sich «Event» tatsächlich im selben Status wie «From» , und zugleich innerhalb der Koordinaten «x1,x2,y» befindet. Die Umwandlung in «To» Keyboard erfolgt danach in «Event».

FIO_1

Einleitung

Dieser Modul stellt Funktionen für Dateioperationen allgemeiner Art zur Verfügung. Die Bildschirmanzeige von Dateien ist in FIO_2 und FIO_3 abgehandelt.

Beschreibung

```
IMPORT FIO, Str;
FROM Com_1 IMPORT tLine, tPath, tName;
FROM FIO IMPORT File, FileAttr, Attrib;
FROM FIOR IMPORT ExtStr;
FROM Tex_1 IMPORT pcText;
FROM Scr_1 IMPORT rPal, cTimer;
FROM Scr_3 IMPORT cPick;

CONST
        EOL = Str.CHARSET{CHR(10),CHR(13)};
TYPE
        rEntry = RECORD
           Name : tName;
           Attr : FileAttr;
           Date : CARDINAL;
           Time : CARDINAL;
           Size : LONGCARD;
        END;
```

PROCEDURE BuildDir(VAR HP:tPath);

Ein Aufruf stellt in «HP» den Pfad zur aktiven Directory zur Verfügung.

PROCEDURE BuildTemp(Pfx,Ext:ExtStr):tPath;

Erstellt mit «Prefix» , «Extension» (beide 1..3-stellig), TEMP-Pfad und HEX-Ticker (DOS_1.GetTicker) die Angaben für eine Temporärdatei.

Z.B: temp = 'C:\TEMP\PFXhhhhh.EXT' => temp:=BuildTemp('pfx','ext');

PROCEDURE ExistsDir(HP:tPath; New:BOOLEAN):CARDINAL;

Prüft, ob die in «HP» angegebene Directory existiert. Mit «New» =TRUE wird sie ggf. erstellt. Als Ergebnis wird der DOS-Returncode geliefert.

PROCEDURE FindFile(VAR FP:ARRAY OF CHAR):BOOLEAN;

Sucht in den PATH-Verzeichnissen von DOS nach der Datei, deren Namen in «FP» steht und ergänzt sie danach mit den Pfadangaben. Die Aktion wird im Funktionswert bestätigt.

PROCEDURE FormPath(VAR Path:tPath);

Formt die Angaben in «Path» gemäss DOS-Standard. Die einzelnen Elemente von «Path» können durch Leerschlag, wie durch Schrägstrich separiert sein. Somit ist z.B. 'c: pfad1 pfad2' ebenso gültig wie 'c:/pfad1 / pfad2'.

PROCEDURE GetLC(VAR FP:ARRAY OF CHAR; VAR Pos:LONGCARD;
VAR Ln,Cn:CARDINAL):BOOLEAN;

Ermittelt in der Datei «FP» Zeilen- und Spaltennummer («Ln»/«Cn») von Position «Pos». Die Aktion wird im Funktionswert bestätigt.

PROCEDURE RdDAT(Module:ARRAY OF CHAR;
VAR DAT:ARRAY OF BYTE):BOOLEAN;

Sucht und liest die beiden Dateien User.DAT und Module.DAT. In «Module» darf keine Dateierweiterung und keine Pfadangabe verwendet werden. User.DAT wird in die Variable SCR_1.User gelesen, Module.DAT (Modul.EXE) an die Stelle von «DAT» . Beide Dateien enthalten Komponenten für die Steuerung der Module. Falls sie fehlen wird deren Speicherbereich intialisiert. Änderungen in der Länge werden automatisch abgehandelt. (Zurückgeschrieben werden sie mit WrDAT). Die Aktion wird im Funktionswert bestätigt.

PROCEDURE RdItem(F:File; Separ,Delete:Str.CHARSET;
VAR Pos:LONGCARD; VAR Item:ARRAY OF CHAR):BOOLEAN;

Liest, bzw. bildet «Items» aus der mit «F» verbundenen Datei. Massgebend für die Itembildung sind «Separatoren», also jene Zeichen, welche die Items trennen, und «Delete», also Zeichen, welche vor der Itembildung zu entfernen sind (z.B: '{}' für Zusammen{ge}hörend). Sobald ein «Item» vollständig ist, wird die Kontrolle zurückgegeben. Trennzeichen am Zeilenende werden automatisch behandelt. «Pos» steuert die Navigation in der Datei. Das Dateiende wird im Funktionswert angezeigt.

PROCEDURE Wildcard(FN:tName):BOOLEAN;

Testet «FN» auf vorhandene Wildcards (*?) Und gibt das Ergebnis im Funktionswert zurück.

PROCEDURE WrDAT(VAR DAT:ARRAY OF BYTE);

Schreibt die beiden Dateien User.DAT und Module.DAT zurück (RdDAT).

```
TYPE
    tAccept = PROCEDURE(tPath,rEntry):BOOLEAN;
    tApply = PROCEDURE(tPath,rEntry);

    CLASS cEntry(cPick,cTimer);
      Beam : ARRAY[0..11] OF CHAR;
      PROCEDURE Flag(Tex:pcText; VAR HP:tPath; MatchFile:ARRAY OF CHAR;
                                 VAR Pal:rPal; Accept:tAccept; Apply:tApply);
      PROCEDURE Pick(Tex:pcText; VAR HP:tPath; MatchFile:ARRAY OF CHAR;
                                 VAR Pal:rPal; Accept:tAccept; Apply:tApply):BOOLEAN;
    END cEntry;
```

Navigiert durch Verzeichnisse um dort Dateien auszuwählen (Pick), oder mit einem Kennzeichen (Flag) zu versehen, und diese danach gesammelt zu verarbeiten. Die Variable Beam definiert die Breite des Auswahlcursors für Scr_3.cPick. Die Prozeduren im Ablauf: **Flag**(Loop) oder **Pick**(Loop).

```
PROCEDURE Flag(Tex:pcText; VAR HP:tPath; MatchFile:ARRAY OF CHAR;
                           VAR Pal:rPal; Accept:tAccept; Apply:tApply);
```

Dient dazu, Dateien für eine bestimmte Aktion auszuwählen, um sie danach gesammelt verarbeiten zu können. Das Bild-Handling wird dabei von Scr_3.cPick übernommen. In «Tex» ist der Pointer auf die Textdaten anzugeben, in «HP» der Ausgangspfad, und in «MatchFile» ein bis mehrere Dateinamen (Jokerzeichen sind erlaubt). Für das Auswahlfenster wird in «Pal» eine Farbpalette verlangt. «Accept» bezeichnet eine Prozedur, welche mit jeder vorselektierten Datei aufgerufen wird, wobei deren Funktionswert dann die Aufnahme der Entry (Datei) in das Auswahlfenster entscheidet. «Apply» bezeichnet jene Prozedur, welche ausgewählte Dateien einzeln verarbeitet. Der Aufruf erfolgt bei beiden Prozeduren mit Pfad und Entry (Datei).

```
PROCEDURE Pick(Tex:pcText; VAR HP:tPath; MatchFile:ARRAY OF CHAR;
                           VAR Pal:rPal; Accept:tAccept; Apply:tApply):BOOLEAN;
```

Dient dazu, eine Datei für eine bestimmte Aktion auszuwählen, um sie danach zu verarbeiten. Das Bild-Handling wird dabei von Scr_3.cPick übernommen. In «Tex» ist der Pointer auf die Textdaten anzugeben, in «HP» der Ausgangspfad, und in «MatchFile» ein bis mehrere Dateinamen (Jokerzeichen sind erlaubt). Für das Auswahlfenster wird in «Pal» eine Farbpalette verlangt. «Accept» bezeichnet eine Prozedur, welche mit jeder vorselektierten Datei aufgerufen wird, wobei deren Funktionswert dann die Aufnahme der Entry (Datei) in das Auswahlfenster entscheidet. «Apply» bezeichnet jene Prozedur, welche die ausgewählte Datei verarbeitet. Der Aufruf erfolgt bei beiden Prozeduren mit Pfad und Entry (Datei). Als Funktionswert von Pick wird die Auswahl bestätigt.

```
    CLASS cPage;
      F : File;
      Left  : LONGCARD;
      Right : LONGCARD;
      Lines : CARDINAL;
      PROCEDURE Close;
      PROCEDURE Open(FP:tPath; ReadOnly:BOOLEAN);
      PROCEDURE RdArrDn(VAR Is:CARDINAL;
                        VAR Pg:ARRAY OF BYTE; Ln:ARRAY OF CHAR):BOOLEAN;
```

```
    PROCEDURE RdArrUp(VAR Is: CARDINAL;
                      VAR Pg:ARRAY OF BYTE; Ln:ARRAY OF CHAR):BOOLEAN;
    PROCEDURE RdLeft(From:LONGCARD; VAR Ln:ARRAY OF CHAR);
    PROCEDURE RdPg(From:LONGCARD; With:CARDINAL; VAR Is:CARDINAL;
                      VAR Pg:ARRAY OF BYTE; Ln:ARRAY OF CHAR):BOOLEAN;
    PROCEDURE RdPgDn(With:CARDINAL; VAR Is:CARDINAL;
                      VAR Pg:ARRAY OF BYTE; Ln:ARRAY OF CHAR):BOOLEAN;
    PROCEDURE RdPgUp(With:CARDINAL; VAR Is:CARDINAL;
                      VAR Pg:ARRAY OF BYTE; Ln:ARRAY OF CHAR):BOOLEAN;
END cPage;
```

Dient dazu, in Dateien vorwärts und rückwärts zu blättern, zeilen- und seitenweise. Die Grösse einer Seite ist grundsätzlich durch «Pg» definiert, kann aber mittels «Lines» eingeschränkt werden. «Pg» muss durch «Ln» ohne Rest teilbar sein, ansonsten wird abgebrochen. Idealerweise wird deshalb für «Ln» eine Zeile von «Pg» verwendet. «With» und «Is» laufen von 0..HIGH(Page) und positionieren den Zeilen-Cursor vor- und nach dem Lesen.

```
PROCEDURE Close;
```

Die mit «F» verbundene Datei wird geschlossen.

```
PROCEDURE Open(FP:tPath; ReadOnly:BOOLEAN);
```

Die Datei «FP» wird im Status gemäss «ReadOnly» geöffnet.

```
PROCEDURE RdArrDn(VAR Is:CARDINAL;
                  VAR Pg:ARRAY OF BYTE; Ln:ARRAY OF CHAR):BOOLEAN;
```

Verschiebt «Pg» um eine Zeile in Richtung Dateiende. Im Ergebnis bezeichnet «Is» die neue Cursorzeile. Im Funktionswert wird «Weiterlesen» angezeigt.

```
PROCEDURE RdArrUp(VAR Is: CARDINAL;
                  VAR Pg:ARRAY OF BYTE; Ln:ARRAY OF CHAR):BOOLEAN;
```

Verschiebt «Pg» um eine Zeile in Richtung Dateianfang. Im Ergebnis bezeichnet «Is» die neue Cursorzeile. Im Funktionswert wird «Weiterlesen» angezeigt.

```
PROCEDURE RdLeft(From:LONGCARD; VAR Ln:ARRAY OF CHAR);
```

Liefert in «Ln» die vollständige Zeile in Relation zur Dateiposition «From».

```
PROCEDURE RdPg(From:LONGCARD; With:CARDINAL; VAR Is:CARDINAL;
                  VAR Pg:ARRAY OF BYTE; Ln:ARRAY OF CHAR):BOOLEAN;
```

Liefert eine «Pg» ab Dateiposition «From» unter Berücksichtigung des Zeilen-Cursors «With». Im Ergebnis bezeichnet «Is» die neue Cursorzeile. Im Funktionswert wird «Weiterlesen» angezeigt.

```
PROCEDURE RdPgDn(With:CARDINAL; VAR Is:CARDINAL;
                  VAR Pg:ARRAY OF BYTE; Ln:ARRAY OF CHAR):BOOLEAN;
```

Verschiebt «Pg» um eine Seite in Richtung Dateiende, unter Berücksichtigung des Zeilen-Cursors «With». Im Ergebnis bezeichnet «Is» die neue Cursorzeile. Im Funktionswert wird «Weiterlesen» angezeigt.

```
PROCEDURE RdPgUp(With:CARDINAL; VAR Is:CARDINAL;
                  VAR Pg:ARRAY OF BYTE; Ln:ARRAY OF CHAR):BOOLEAN;
```

Verschiebt «Pg» um eine Seite in Richtung Dateianfang, unter Berücksichtigung des Zeilen-Cursors «With». Im Ergebnis bezeichnet «Is» die neue Cursorzeile. Im Funktionswert wird «Weiterlesen» angezeigt.

FIO_2

Einleitung

Dieser Modul realisiert Funktionen zur Bildschirmausgabe von Dateien. Pro Variable der Klasse cView kann eine beliebige Anzahl von Dateien der Klasse cFile abgehandelt werden. Diese befinden sich - bildlich gesprochen - auf einer Ebene und können wechselseitig in den Vordergrund gebracht werden. Auch Hypertexteigenschaften werden unterstützt.

Beschreibung

```
FROM Com_1 IMPORT tLine;
FROM FIO_1 IMPORT tPath;
FROM Scr_1 IMPORT sKb, prPal, RelCoord, WinType, cMap;
FROM Tab_2 IMPORT cTab2, cEle2, pcEle2;
FROM Tex_1 IMPORT pcText;

TYPE
        prFile;
        pPath = POINTER TO tPath;
        pcFile = POINTER TO cFile;

        CLASS cFile(cEle2);
          Nr : CARDINAL; (* for Compare *)
          Path : pPath;
          FD : prFile;
          VIRTUAL PROCEDURE Compare(p:pcEle2):INTEGER;
          VIRTUAL PROCEDURE Delete(p:pcEle2);
          VIRTUAL PROCEDURE ExitBrackets(p:pcEle2; VAR Beam:ARRAY OF CHAR):BOOLEAN;
          VIRTUAL PROCEDURE ExitLine(p:pcEle2; VAR Line:ARRAY OF CHAR):BOOLEAN;
        END cFile;
```

Bildet die Schnittstelle zu cView für alle ihre Dateien. Von den virtuellen Prozeduren können Delete, ExitBrackets (Hypertext) und ExitLine bei Bedarf überschrieben werden.

```
        VIRTUAL PROCEDURE Compare(p:pcEle2):INTEGER;
```

Dient der Einordnung von Dateien in die interne Tabelle. Diese Prozedur soll nicht überschrieben werden.

```
        VIRTUAL PROCEDURE Delete(p:pcEle2);
```

Diese Prozedur wird aufgerufen, bevor eine Datei endgültig verlassen wird.

```
        VIRTUAL PROCEDURE ExitBrackets(p:pcEle2; VAR Beam:ARRAY OF CHAR):BOOLEAN;
```

Diese Prozedur wird aufgerufen, wenn auf einem mit {Hypertext} markierten Bereich [Enter] betätigt wurde. In «Beam» wird der Hypertext geliefert. Der Funktionswert steuert die Rückgabe der Kontrolle in cView.View an die aufrufende Instanz, damit dort eine neue Datei installiert werden kann.

```
        VIRTUAL PROCEDURE ExitLine(p:pcEle2; VAR Line:ARRAY OF CHAR):BOOLEAN;
```

Diese Prozedur wird aufgerufen, wenn auf einer Stelle ohne Hypertext [Enter] betätigt wurde. In «Line» wird die Zeile geliefert. Der Funktionswert steuert die Rückgabe der Kontrolle in cView.View an die aufrufende Instanz, damit dort eine bestimmte Aktion stattfinden kann.

```
TYPE
        prView;

        CLASS cView(cTab2);
          VD : prView;
          PROCEDURE Add(VAR File:cFile; FP:tPath; Pos:LONGCARD; Tex:pcText;
                        ViewPal,FindPal:prPal; Title,L25:ARRAY OF CHAR;
                        LineBody:CARDINAL;
                        Scale,Brackets:BOOLEAN; Break:CHAR):CARDINAL;
          PROCEDURE Delete(VAR File:cFile);
```

```
    PROCEDURE GetLine(VAR Line:ARRAY OF CHAR);
    PROCEDURE GotoPos(Pos:LONGCARD);
    PROCEDURE PutClue(Clue:ARRAY OF CHAR);
    PROCEDURE PutError(Error:ARRAY OF CHAR);
    PROCEDURE View(Return:sKb; VAR File:pcFile; VAR Kb:CHAR);
END cView;
```

Steuert den Ablauf im Kontext mit der Ausgabe von Dateien am Bildschirm. Die wichtigsten Prozeduren im Ablauf sind: **Add** und **View**(Loop).

```
PROCEDURE Add(VAR File:cFile; FP:tPath; Pos:LONGCARD; Tex:pcText;
                                  ViewPal,FindPal:prPal; Title,L25:ARRAY OF CHAR;
                                  LineBody:CARDINAL;
                                  Scale,Brackets:BOOLEAN; Break:CHAR):CARDINAL;
```

Installiert die Datei «FP» mit den Daten «File» in der Klasse. Mit «Pos» 0..MAX(LONGCARD) wird die Startposition festgelegt. «Tex» ist der Pointer auf die Textdaten, «ViewPal» und «FindPal» auf die entsprechenden Farbpaletten. «Title» ist die Bildüberschrift und «L25» der Inhalt von Zeile 25. «LineBody» bezeichnet den sichtbaren Ausschnitt einer Zeile. Bei unter- bzw. überschreiten von Limiten wird er angepasst. «Scale» steuert die Anzeige eines dynamischen Rasters, «Brackets» die Reaktion auf Hypertext und «Break» die Trennlinie bei Druckseiten (wenn «Break» in der Datei angetroffen wird). Als Funktionswert wird die Dateireferenz (cFile.Nr), bzw. MAX(CARDINAL) im Fehlerfall, zurückgegeben.

```
PROCEDURE Delete(VAR File:cFile);
```

Entfernt die Datei gemäss «File» .Nr aus der Klasse. Bei [Esc] oder [AltX] geschieht dies automatisch.

```
PROCEDURE GetLine(VAR Line:ARRAY OF CHAR);
```

Liefert den Inhalt der Zeile, auf der der Bild-Cursor positioniert ist.

```
PROCEDURE GotoPos(Pos:LONGCARD);
```

Verschiebt die Bildausgabe an die Dateiposition «Pos».

```
PROCEDURE PutClue(Clue:ARRAY OF CHAR);
```

Schreibt «Clue» als Hinweis in die Zeile 24.

```
PROCEDURE PutError(Error:ARRAY OF CHAR);
```

Schreibt «Error» als Fehler in die Zeile 24.

```
PROCEDURE View(Return:sKb; VAR File:pcFile; VAR Kb:CHAR);
```

Bringt die Datei gemäss «File».Nr in den Vordergrund. In «Return» können zusätzliche Funktionstasten festgelegt werden, bei denen View die Kontrolle wieder abzugeben hat, was auch von cFile.ExitBrackets und cFile.ExitLine ausgelöst sein kann. Die auslösende Funktionstaste steht in «Kb».

FIO_3

Einleitung

Dieser Modul realisiert Funktionen zur Bildschirmausgabe von Dateien im Full-Screen-Modus, mit und ohne Überschriftten.

.

Beschreibung

```
FROM FIO_1 IMPORT cPage, tPath;
FROM Scr_1 IMPORT sKb, prPal, cMap;
FROM Tex_1 IMPORT pcText;

TYPE
        tPgLn = ARRAY[0..1999]OF CHAR;
        prFind;
        pcFullScr = POINTER TO cFullScr;

        CLASS cFullScr(cMap,cPage);
          page : ARRAY[0..19]OF tPgLn;
          more : BOOLEAN;
          isPg,maxPg : CARDINAL;
          area1,area2 : ADDRESS;
          xChr : CARDINAL;
          XChr : CARDINAL;
          heads : CARDINAL;
          find : prFind;
          PROCEDURE Close;
          PROCEDURE Open(FP:tPath; Pos:LONGCARD; Frame:BOOLEAN; Heads:CARDINAL;
                                   Tex:pcText; ViewPal,FindPal:prPal;
                                   Title,L25:ARRAY OF CHAR):BOOLEAN;
          PROCEDURE GotoPos(Pos:LONGCARD);
          PROCEDURE SetHead(Nr:CARDINAL; VAR Head:ARRAY OF CHAR);
          PROCEDURE View(Return:sKb; VAR Kb:CHAR);
        END cFullScr;
```

Die wichtigsten Prozeduren im Ablauf sind: **Open**, **SetHead**, **View**(Loop), **Close**.

```
        PROCEDURE Close;
```

Schliesst die Datei.

```
        PROCEDURE Open(FP:tPath; Pos:LONGCARD; Frame:BOOLEAN; Heads:CARDINAL;
                                 Tex:pcText; ViewPal,FindPal:prPal;
                                 Title,L25:ARRAY OF CHAR):BOOLEAN;
```

Öffnet die Datei gemäss «FP». Mit «Pos» 0..MAX(LONGCARD) wird die Startposition festgelegt. «Frame» steuert die Verwendung eines Rahmens. «Heads» reserviert die Zeilen für Überschriften. «Tex» ist der Pointer auf dieTextdaten, «ViewPal» und «FindPal» auf die entsprechenden Farbpaletten. «Title» ist die Bildüberschrift und «L25» der Inhalt von Zeile 25. Die Aktion wird im Funktionswert bestätigt.

```
        PROCEDURE GotoPos(Pos:LONGCARD);
```

Verschiebt die Bildausgabe an die Dateiposition «Pos».

```
        PROCEDURE SetHead(Nr:CARDINAL; VAR Head:ARRAY OF CHAR);
```

Installiert die Überschrift in der Zeile gemäss «Nr» .

```
        PROCEDURE View(Return:sKb; VAR Kb:CHAR);
```

Zeigt den Inhalt der Datei an. In «Return» enthaltene Funktionstasten steuern die Abgabe der Kontrolle an die aufrufende Instanz. In «Kb» wird die auslösende Funktionstaste zürckgegeben.

Hlp_1

Einleitung

Dieser Modul realisiert das dynamische Hilfesystem mit Hypertexteigenschaften. Es benötigt wenig Speicherplatz, da es im Prinzip einzig Referenzen auf die Hilfetexte verwaltet, dies aber nur für aktuelle Belange. Die Texte können mit einem beliebigen Editor für ASCII-Dateien gepflegt werden.

Die Definitionen in der Hilfedatei folgen dem Prinzip von Prefix und Suffix. An der Stellung des Dachsymbols erkennt man Prefix^ und ^Suffix. Zusammen bilden sie den Schlüssel (Key) für den Zugriff auf die Daten. Dabei findet keine Differenzierung nach Gross-/Kleinschrift und Umlautschreibweise statt. Zeilen ohne Prefix- und Suffix-Symbole sind Hilfetext. Das Tabulatorzeichen CHR(9) wird korrekt abgehandelt. Eine Zeile darf maximal 256 Stellen belegen, darüber wird abgeschnitten. Die Beispiele sind jeweils mit ♣ markiert.

1. Dateinamen

Der Name muss so festgelegt werden, dass er (a) mit keiner bestehenden Datei *.HP* kollidiert und (b) als Dateiname vom DOS akzeptiert wird. Der Namensgeber-Prefix darf länger als der Dateiname sein. Aus «Beispielsweise^» entsteht somit die Datei «BEISPIEL.HP*», wobei * für die erste Stelle der Sprache steht (English,French,German,Italy,Spanish).

2. Prefix-Zeile

Der Prefix ist ein sprachunabhängiger Schlüssel. Alle nachfolgend definierten Suffixe sind ihm untergeordnet, bis ein weiterer Prefix folgt. In der Datei wird der Prefix vollständig angegeben: ♣ Beispielsweise^
In welcher Spalte er beginnt ist unwesentlich. Nach einem Leerschlag kann eine Überschrift für das Index-Fenster folgen. In einer Datei dürfen mehrere Prefixe stehen. Der Dateiname muss jedoch vom Hauptprefix abgeleitet sein. Die andern sind versteckte Schlüssel und lassen sich nur indirekt finden. So muss z.B. in einer Modul-Hierarchie die höchste Ebene innerhalb der Datei als Namensgeber fungieren. Das Prinzip beim Suchen ist dabei folgendes: Ein dem System unbekannter Prefix wird als Datei gesucht. Ist die Datei gefunden, werden alle weiteren Prefixe in dieser Datei adressiert und sind damit bekannt.

```
♣ Beispielsweise^  Überschrift
          ^Suffix
    Hilfetext
          ...
    Versteckt^ ...
```

3. Suffix-Zeile

Der Suffix ist ein sprachunabhängiger Schlüssel. Er ist dem vorangehenden Prefix untergordnet. Die Zeile besteht aus dem Suffix und, nach einem Leerschlag, einer Überschrift. In welcher Spalte der Suffix beginnt ist unwesentlich. Die Überschrift lässt beide zu Indizes werden, welche im Text als Verweis dienen können (Punkt 4). In dem Zusammenhang sei auch auf die Bildung von Synonymen hingewiesen (Punkt 5). Auf die Zeile mit dem Suffix folgt der Hilfetext, der mindestens eine Zeile belegen muss. Ansonsten wird die Suffix-Zeile ignoriert.

```
♣ ^Suffix Überschrift
      Hilfetext
```

4. Hilfetext

Mit dem Hilfetext darf in einer beliebigen Spalte begonnen werden. Alle Zeilen eines Suffix werden vom Hilfesystem automatisch auf jene mit der tiefsten Spalte ausgerichtet. Will man auf einen anderen Text Bezug nehmen ist das Schlagwort in geschweifte Klammern zu schreiben ♣ {Hypertext}. Es hebt sich damit farblich vom übrigen Text ab. Existiert dieses Schlagwort innerhalb eines Prefix als Index (es geht also aus Suffix/Überschrift hervor), ist keine weitere Aktion notwendig. In allen anderen Fällen muss die Verbindung als Synonym definiert werden (Punkt 5).

5. Synonym

Ein Synonym stellt die Verbindung zu existierenden Indizes her und kann auf einen beliebigen Prefix in einer beliebigen Datei verweisen.

♣ ^Synonym=Index *(für Verbindungen innerhalb des Prefix)*
♣ ^Synonym=Prefix.Suffix *(für Verbindungen ausserhalb des Prefix)*

Beschreibung

```
FROM FIOR IMPORT ExtStr;
FROM Scr_1 IMPORT prPal;
FROM Tex_1 IMPORT pcText;

TYPE
        prHelp;
        pcHelp = POINTER TO cHelp;

        CLASS cHelp;
           HD : prHelp;
           PROCEDURE Close;
           PROCEDURE Index(Key:ARRAY OF CHAR):BOOLEAN;
           PROCEDURE Open(File:ARRAY OF CHAR; Tex:pcText;
                                IndexPal,ViewPal,FindPal:prPal):BOOLEAN;
           PROCEDURE View(Key:ARRAY OF CHAR):BOOLEAN;
        END cHelp;
```

Die Prozeduren im Ablauf sind: **Open**, **Index**(Loop) oder **View**(Loop), **Close**(nur für Spezialfall).

```
VAR
        Help : cHelp;
```

```
        PROCEDURE Close;
```

Schliesst das Hilfesystem und gibt die Ressourcen frei.

```
        PROCEDURE Index(Key:ARRAY OF CHAR):BOOLEAN;
```

Zeigt den Index gemäss«Key» an, bzw. alle Indizes, wenn «Key» leer ist. Von dort aus verzweigt [Enter] auf View. Im Funktionswert wird die Aktion bestätigt.

```
        PROCEDURE Open(File:ARRAY OF CHAR; Tex:pcText;
                          IndexPal,ViewPal,FindPal:prPal):BOOLEAN;
```

Startet das Hilfesystem mit «File» als Einstiegspunkt (i.d.R. Modul.HP* = Modul.EXE). «Tex» ist der Pointer auf die Textdaten, «IndexPal» und «ViewPal» auf die entsprechenden Farbpaletten.

```
        PROCEDURE View(Key:ARRAY OF CHAR):BOOLEAN;
```

Zeigt den Hilfetext gemäss«Key» an. Im Funktionswert wird die Aktion bestätigt.

Mem_1

Einleitung

Dieser Modul stellt Funktionen für die dynamische Speicherverwaltung zur Verfügung. Er dient als Bindeglied zu Modula-2, damit in den Prozeduren auf die Meldung, dass zuwenig Speicher verfügbar ist, situativ reagiert werden kann (z.B. mit User.Flush). Allozierter Speicher ist immer mit Null initialisiert.

Beschreibung

PROCEDURE ALLOCATE(VAR a:ADDRESS; s:CARDINAL);
Alloziert Speicher von «s» Byte an die Adresse «a».

PROCEDURE Available(s:CARDINAL):BOOLEAN;
Testet auf verfügbaren Speicher von «s» Byte. Im Funktionswert wird der Sachverhalt bestätigt.

PROCEDURE DEALLOCATE(VAR a:ADDRESS; s:CARDINAL);
Dealloziert Speicher von «s» Byte an der Adresse «a».

PROCEDURE DISPOSE(VAR a:ADDRESS; VAR b:ARRAY OF BYTE);
Dealloziert Speicher in der Länge von «b» Byte an der Adresse «a».

PROCEDURE NEW(VAR a:ADDRESS; b:ARRAY OF BYTE);
Alloziert Speicher in der Länge von «b» Byte an die Adresse «a».

Scr_1

Einleitung

Dieser Modul realisiert Funktionen für die Ein- und Ausgabe von Daten am Bildschirm. In Scr_2 ist die Benutzerorientierung, in Scr_3 eine Klasse zur Auswahl von Daten, und in Scr_4 die Kontrolle über alle Fenster realisiert.

Koordinaten sind definiert als (a) absolut: X,Y = 0,0..79,24 und (b) relativ: x,y = 1,1..80,25. Funktionstasten werden jeweils in eckigen Klammern [ ] dargestellt, wobei z.B. [AltX] gleichzeitiges betätigen von [Alt] und [X] meint. *Kursiv dargestellte Tasten lassen sich nicht beeinflussen und sind deshalb in keiner Return-Menge sinnvoll.* Auf den Fehler- und Hinweiszeilen (24, 25) verhält sich die Maus textsensitiv. Damit lassen sich die dort aufgeführten Funktionen durch 1xMausKlick starten. Die Mausoptionen können den persönlichen Bedürfnissen anpasst werden.

Abbrechen
[Esc], [AltX], 1xMausCancel

Auswahlfelder der Map
[SpaceBar], [F5], 2xMausKlick

Balken-/Cursorposition
[Home], [End], [AltHome], [AltEnd], [Tab], [ShiftTab], [ArrUp], [ArrDn], [ArrLt], [ArrRt], [PgUp], [PgDn], *1xMausKlick*

Dateneingabe
[Ins], [Del], [CtrlEnd], [BackSpace], [AltBackSpace]

Enter, Starten
[↵], 2xMausKlick, [F7]

Fensterposition/-grösse
[CtrlPgUp], [CtrlPgDn], [CtrlArrUp], [CtrlArrDn], [CtrlArrLt], [CtrlArrRt], [F5]

Hilfe
[F1], [AltF1]

Setup (Farbgebung)
[AltS]

Textsuche
[AltF], [AltTab]

Beschreibung

```
FROM Com_1 IMPORT sPict, tPath;
FROM Dat_1 IMPORT rDate;
FROM Eve_1 IMPORT rMouse;
FROM Tex_1 IMPORT tText, tPos, pcText;
FROM Window IMPORT AbsCoord, RelCoord, PaletteDef, PaletteColorDef, PaletteRange, TitleMode,
        WhereX, WhereY, WinType;
```

Keyboard

```
CONST
        BackSp = CHR(8);    CtrlC = CHR(3);    CtrlEnter = CHR(10);          (* AsciiKeys *)
        Enter  = CHR(13);   Esc   = CHR(27);   TabRt     = CHR(9);

        AltBackSp = CHR(14);                                                  (* ExtendedKeys *)
        AltEnd = CHR(159);          AltS    = CHR(31);
        AltF = CHR(33);             AltTab  = CHR(165);
```

```
        AltF1= CHR(104);        AltHome = CHR(151);
        AltX = CHR(45);
        ArrLt = CHR(75);        ArrRt = CHR(77);
        ArrUp = CHR(72);        ArrDn = CHR(80);
        CtrlArrLt = CHR(115);   CtrlArrRt = CHR(116);
        CtrlArrUp = CHR(141);   CtrlArrDn = CHR(145);
        CtrlDel = CHR(147);
        CtrlPgUp = CHR(118);    CtrlPgDn = CHR(132);
        F1  = CHR(59);          F2  = CHR(60);          F3  = CHR(61);
        F10 = CHR(68);          F11 = CHR(133);         F12 = CHR(134);
        F4  = CHR(62);          F5  = CHR(63);          F6  = CHR(64);
        F7  = CHR(65);          F8  = CHR(66);          F9  = CHR(67);
        Ins = CHR(82);          Del = CHR(83);
        PgUp = CHR(73);         PgDn = CHR(81);
        Home = CHR(71);         End = CHR(79);
        TabLt = CHR(15);
TYPE
        sKb = SET OF CHAR;
```

Anbindung der Fenster

```
TYPE
        rLink = RECORD
           To : WinType;
           PosXY  : BOOLEAN;
           CursOn : BOOLEAN;
        END;
```

Farbgebung für Fenster (Paletten)

```
CONST
        Ground0 = 0;    Frame1 = 1;     Beam2 = 2;      Brack3 = 3;  (* only for L1 and PopUp *)
        Enter3 = 3;     Skip4 = 4;      Action5 = 5;    Choice6 = 6;
        Hyper7 = 7;     Hyper8 = 8;
TYPE
        tPalType = (TypeL1,TypePop,TypeBasic,TypeMap);
        sPalType = SET OF tPalType;

        prPal = POINTER TO rPal;
        rPal = RECORD
           Type : sPalType;
           Base,L24,L25 : PaletteDef;
           ReservedFor : ARRAY[0..19]OF CHAR;
        END;

        rStandardPal = RECORD
           L1,Pop,Basic,Map : rPal;
        END;

        rHelpPal = RECORD
           Index,View,Find : rPal;
        END;
```

Benutzerorientierung des Systems

```
TYPE
        tDelayFact = [1..9];

        rBool = RECORD
           Item : CARDINAL;
           True : CHAR;
           False : CHAR;
        END;
```

```
rSPF = RECORD
   Prg : tPath;
   Par : tPath;    (* @File, @Word *)
   Pal : rPal;
END;

tDateFormat = (MDY,DMY,YMD);
rDateFormat = RECORD
   Format : tDateFormat;
   Mark : CHAR;
END;

rUser = RECORD
   Ref : ARRAY[0..4] OF CHAR;
   Pal : rStandardPal;
   Flag : CHAR;
   Lang : CHAR;
   Delay : tDelayFact;
   State : CARDINAL;
   Bool : rBool;
   Sound : BOOLEAN;
   Help : rHelpPal;
   Choice: rPal;
   SPF : rSPF;
   Date : rDateFormat;
   Wait : CARDINAL;
   Mouse : rMouse;
   Head  : rPal;
END;
```

Globale Variablen

```
VAR
   AltSOn : BOOLEAN;
   AltXOn : BOOLEAN;
   AltXProc : PROC;
   AppMod : ARRAY[0..7] OF CHAR;
   AppPath : tPath;
   ChoiceKb : CHAR;
   DelayL24 : tText;
   EscOn : BOOLEAN;
   ExitL24 : tText;
   HelpL24 : tText;
   PendL24 : tText;
   StopL24 : tText;
   User : rUser;
   ValueL24 : tText;
   WaitL24 : tText;
```

Globale Prozeduren

PROCEDURE AltXEsc;

Wenn diese Prozedur in AltXProc installiert ist werden die Tasten [AltX] und [Esc] als «gleichwertig» behandelt und pendent gehalten, bis in AltXProc die Prozedur AltXHalt installiert ist. Dies sichert die Integrität bestimmter Verarbeitungen.

PROCEDURE AltXHalt;

Wenn diese Prozedur in AltXProc installiert ist führt [AltX] sofort zum Abbruch der Verarbeitung (ist Standard)

PROCEDURE AltXWait;

Wenn diese Prozedur in AltXProc installiert ist wird [AltX] pendent gehalten, bis die Prozedur AltXHalt installiert wird. Dies sichert die Integrität bestimmter Verarbeitungen.

PROCEDURE GetChoiceBody(VAR Text:ARRAY OF CHAR; Max:CARDINAL):CARDINAL;
Liefert im Funktionswert die optimale Länge in Bezug auf die (Komma)Items von «Text», unter Berücksichtigung des gegebenen Maximums in «Max». Damit wird der Fensterausschnitt bei Auswahlfeldern festgelegt.

PROCEDURE GetColor(Title:ARRAY OF CHAR;VAR Get:PaletteColorDef; VAR Pal:rPal);
Verzweigt in das Fenster zur Farbauswahl und zeigt dort «Title» im Kopf des Fensters an. «Pal» ist die Palette für das Auswahlfenster. Die dort ausgewählte Farbe wird in «Get» zurückgegeben.

PROCEDURE GetDateMask(VAR Mask:ARRAY OF CHAR);
Liefert in «Mask» die vom Benutzer gewünschte Darstellung des Datums. Die Maske ist in Dat_1 behandelt.

PROCEDURE GetPal(For:ARRAY OF CHAR; VAR Pal:ARRAY OF rPal):prPal;
Liefert im Funktionswert den Pointer auf die zutreffende Farbpalette. In «For» muss der Name des Fensters stehen. Dieser wird in «Pal» gesucht und, wenn noch nicht vorhanden, dort eröffnet (Überlauf ist HIGH(Pal)!).

PROCEDURE InitGlobal(Tex:pcText; Help:ARRAY OF CHAR);
Installiert die globalen Variablen. «Tex» ist der Pointer auf die Textdaten und «Help» der Name für den Start des Hilfesystems (Hlp_1).

PROCEDURE NewColor(VAR Base,L24,L25:WinType; VAR Pal:rPal);
Bewirkt, dass ein Fensterpacket «Base,L24,L25» die neue Farbpalette in «Pal» benutzt. Für fehlende Fensterpointer (WinType) muss NIL übergeben werden.

PROCEDURE PutBrackets(Line:ARRAY OF CHAR);
Schreibt den Inhalt von «Line» im aktuellen Fenster an die aktuelle Position (x,y). Dabei wird Text in geschweiften Klammern {Beispiel} farblich speziell behandelt, die Klammern selbst aber nicht geschrieben.

PROCEDURE PutNormal(Line:ARRAY OF CHAR);
Schreibt den Inhalt von «Line» im aktuellen Fenster an die aktuelle Position (x,y).

PROCEDURE ReplyKey(L24:ARRAY OF CHAR; Return:sKb; VAR Pal:rPal):CHAR;
Schreibt den Inhalt von «L24» in die Zeile 24 mit der Farbpalette «Pal». Ist «Return» eine leere Menge, führt ein beliebiges Ereignis zur Rückkehr, andernfalls nur eines von «Return». Im Funktionswert wird das Ereignis geliefert.

PROCEDURE Sound(FreqHz,Hsecs:CARDINAL);
Generiert einen Ton mit der Frequenz von «FreqHz» (Hertz) in der Dauer von «Hsecs» (1/100 sec).

Array-Fenster

```
TYPE
        prArray;

        CLASS cArray;
          Array : prArray;
          PROCEDURE Add(Text:ARRAY OF CHAR; Flag:BOOLEAN);
          PROCEDURE ChangePal;
          PROCEDURE Clear;
          PROCEDURE ClearBeam;
          PROCEDURE Close;
          PROCEDURE GotoArray(Nr:CARDINAL);
          PROCEDURE Hide;
          PROCEDURE Open(BodyX,NrX,NrY:CARDINAL; Frame:BOOLEAN;
                                Title,Clue:ARRAY OF CHAR; VAR Pal:rPal; Tex:pcText);
          PROCEDURE Pick(Return:sKb; VAR Kb:CHAR;VAR KbEntry:ARRAY OF CHAR):CARDINAL;
          PROCEDURE PutClue(Clue:ARRAY OF CHAR);
          PROCEDURE PutError(Error:ARRAY OF CHAR);
          PROCEDURE PutOnTop;
          PROCEDURE SetLinkPosXY(Flag:BOOLEAN);
          PROCEDURE SetFlag(Nr:CARDINAL; Flag:BOOLEAN);
          PROCEDURE SetMode(Title:ARRAY OF CHAR; Mode:TitleMode);
          PROCEDURE SetOkMsg(Key:CHAR; Msg:ARRAY OF CHAR);
          PROCEDURE SetPage(Dn,Up:BOOLEAN);
          PROCEDURE Setup;
```

END cArray;

Diese Klasse realisiert die Präsentation der Daten in Reihen und Spalten. Bei mehrspaltigem Aufbau läuft die Darstellung von links nach rechts und von oben nach unten. Die Kommunikation mit der Klasse findet aber immer so statt, als wäre nur eine Spalte definiert. Die wichtigsten Prozeduren im Ablauf sind: **Open**, **Add**, **SetOkMsg**, **Pick**(Loop), **Close**.

PROCEDURE Add(Text:ARRAY OF CHAR; Flag:BOOLEAN);
Installiert ein Element «Text» in die nächste freie Position (x,y). «Flag» steuert die Anzeige des benutzerorientierten Selektionsmerkmals.

PROCEDURE ChangePal;
Bewirkt, dass das Fensterpacket die neue Farbpalette in «Pal» benutzt.

PROCEDURE Clear;
Entfernt alle Elemente aus dem Fenster.

PROCEDURE ClearBeam;
Löscht den Cursor-Balken und schaltet ihn aus..

PROCEDURE Close;
Schliesst das Fenster.

PROCEDURE GotoArray(Nr:CARDINAL);
Stellt den Cursor-Balken auf das Element von «Nr» .

PROCEDURE Hide;
Macht das Fenster unsichtbar.

PROCEDURE Open(BodyX,NrX,NrY:CARDINAL; Frame:BOOLEAN;
Title,Clue:ARRAY OF CHAR; VAR Pal:rPal; Tex:pcText);
Öffnet ein Fenster mit Spaltenbreite «BodyX», -zahl «NrX» und Reihenzahl «NrY» . «Frame» steuert die Verwendung eines Rahmens. In «Pal» wird die Farbpalette und in «Tex» der Pointer auf Textdaten übergeben. «Titel» wird in die Kopfmitte des Rahmens (sofern vorhanden), und «Clue» in Zeile 25 geschrieben.

PROCEDURE Pick(Return:sKb; VAR Kb:CHAR;
VAR KbEntry:ARRAY OF CHAR):CARDINAL;
Startet das Auswahlprozedere. Es behält die Kontrolle solange, bis eine Auswahl getroffen ist, ein Ereignis mit «Return» übereinstimmt, oder ein anderes Zeichen (kein Leerschlag) eingegeben wurde. Letztere werden in «KbEntry» angesammelt, bis ein Leerschlag eingeben wird. Die Auswahl ist getroffen, wenn der Funktionswert nicht Null ist.

PROCEDURE PutClue(Clue:ARRAY OF CHAR);
Schreibt «Clue» in Zeile 25 und installiert, bzw. löscht sie.

PROCEDURE PutError(Error:ARRAY OF CHAR);
Schreibt «Error» in Zeile 24 und installiert, bzw. löscht sie.

PROCEDURE PutOnTop;
Sorgt dafür, dass das Fenster zuoberst, damit aktiv und voll sichtbar ist.

PROCEDURE SetLinkPosXY(Flag:BOOLEAN);
Die Fenster werden beim öffnen automatisch an den Vorgänger gelinkt und damit auf dem Bildschirm positioniert. Mit TRUE in «Flag» wird der Linkvorgang aktualisiert, mit FALSE ausgeklinkt.

PROCEDURE SetFlag(Nr:CARDINAL; Flag:BOOLEAN);
Beim Element mit Index «Nr» wird mit TRUE im «Flag» das benutzerorientierte Selektionsmerkmal gesetzt , und mit FALSE gelöscht.

PROCEDURE SetMode(Title:ARRAY OF CHAR; Mode:TitleMode);
Schreibt den Inhalt von «Titel» im Rahmen an die Stelle gemäss «Mode» (NoTitle,LeftUpperTitle, CenterUpperTitle,RightUpperTitle,LeftLowerTitle,CenterLowerTitle,RightLowerTitle).

PROCEDURE SetOkMsg(Key:CHAR; Msg:ARRAY OF CHAR);
Installiert «Msg» und «Key» als akzeptierten Auslöser einer Aktion mit den ausgewählten Daten. Sie werden automatisch aktiviert und deaktiviert, abhängig davon, ob «Key» in Return (Pick) enthalten ist, oder nicht.

PROCEDURE SetPage(Dn,Up:BOOLEAN);
Aktiviert oder sperrt die Tasten fürs Blättern (Arr, Pg) in die gewünschte Richtung «Dn» und «Up».

PROCEDURE Setup;
Verzweigt mit dem Fensterpacket zur Farbgebung.

Basis-Fenster

```
TYPE
    prBasic;

    CLASS cBasic;
      Basic: prBasic;
      PROCEDURE AcceptKey;
      PROCEDURE ChangePal;
      PROCEDURE Clear;
      PROCEDURE ClearBeam;
      PROCEDURE Close;
      PROCEDURE Delay;
      PROCEDURE GetBeam(x1,x2,y:RelCoord);
      PROCEDURE GetBeamCh(VAR Buff:ARRAY OF CHAR);
      PROCEDURE GetChar(Return:sKb; VAR Kb:CHAR;
                                VAR InsOn:BOOLEAN; VAR Get:ARRAY OF CHAR;
                                VAR StrX,WinX,WinY:RelCoord);
      PROCEDURE Hide;
      PROCEDURE MoveCursor(x1,y1,x2,y2:RelCoord;Return:sKb; VAR Kb:CHAR);
      PROCEDURE Open(X1,Y1,X2,Y2:AbsCoord; Cursor,Beam,Frame:BOOLEAN;
                             Title,Clue:ARRAY OF CHAR; VAR Pal:rPal; Tex:pcText);
      PROCEDURE PutChar(Put:ARRAY OF CHAR; PutX,WinX,WinY:RelCoord);
      PROCEDURE PutClue(Clue:ARRAY OF CHAR);
      PROCEDURE PutError(Error:ARRAY OF CHAR);
      PROCEDURE PutOnTop;
      PROCEDURE SetCursor(Flag:BOOLEAN);
      PROCEDURE SetLinkPosXY(Flag:BOOLEAN);
      PROCEDURE SetMode(Title:ARRAY OF CHAR; Mode:TitleMode);
      PROCEDURE Setup;
    END cBasic;
```

Diese Klasse realisiert ein Fenster ohne Spezialitäten. Es ist im Aufbau zugleich Grundlage für die übrigen Fenster. Die wichtigsten Prozeduren im Ablauf sind: **Open**, **PutChar**, **MoveCursor**(Loop), **GetChar**(Loop), **Close**.

PROCEDURE AcceptKey;
Prüft auf [AltS], [AltX] und [Esc] und zeigt in Zeile 24 an, dass die gedrückte Taste akzeptiert ist, aber erst später behandelt wird.

PROCEDURE ChangePal;
Bewirkt, dass das Fensterpacket die neue Farbpalette in «Pal» benutzt.

PROCEDURE Clear;
Entfernt alle Elemente aus dem Fenster.

PROCEDURE ClearBeam;
Löscht den Cursor-Balken und schaltet ihn aus.

PROCEDURE Close;
Schliesst das Fenster.

PROCEDURE Delay;
Verzögert die Programmausführung und behandelt dabei [AltEnd], [AltHome], [+] und [-], welche die Verzögerung entsprechend beeinflussen. Faktor und Startoption können vom Benutzer angepasst werden.

PROCEDURE GetBeam(x1,x2,y:RelCoord);
Generiert den Cursor-Balken im Bereich der angegebenen Koordinaten.

PROCEDURE GetBeamCh(VAR Buff:ARRAY OF CHAR);
Liefert den Inhalt des aktiven Cursor-Balkens.

PROCEDURE GetChar(Return:sKb; VAR Kb:CHAR;
VAR InsOn:BOOLEAN; VAR Get:ARRAY OF CHAR;
VAR StrX,WinX,WinY:RelCoord);
Behandelt die Dateneingabe mit den relativen Startkoordinaten «WinX»,«WinY» bezüglich Fenster und «StrX» bezüglich dem Inhalt von «Get». Falls «Return» eine leere Menge ist, wird die Kontrolle nach jeder Eingabe abgegeben, ansonsten dann, wenn das Ereignis darin enthalten ist. In «Kb» wird das Ereignis geliefert und die Koordinaten sind nachgeführt.

PROCEDURE Hide;
Macht das Fenster unsichtbar.

PROCEDURE MoveCursor(x1,y1,x2,y2:RelCoord;Return:sKb; VAR Kb:CHAR);
Erlaubt die Bewegung des Cursors innerhalb der angegeben Koordinaten. Falls «Return» eine leere Menge ist, wird die Kontrolle nach jeder Eingabe abgegeben, ansonsten dann, wenn das Ereignis darin enthalten ist. In «Kb» wird das Ereignis geliefert.

PROCEDURE Open(X1,Y1,X2,Y2:AbsCoord; Cursor,Beam,Frame:BOOLEAN;
Title,Clue:ARRAY OF CHAR; VAR Pal:rPal; Tex:pcText);
Öffnet ein Fenster in den angegebenen Koordinaten. «Cursor» , «Beam» und «Frame» steuern die entsprechenden (Cursor, Cursor-Balken und Rahmen) Belange. In «Pal» wird die Farbpalette und in «Tex» der Pointer auf Textdaten übergeben. «Titel» wird in die Kopfmitte des Rahmens (sofern vorhanden), und «Clue» in Zeile 25 geschrieben.

PROCEDURE PutChar(Put:ARRAY OF CHAR; PutX,WinX,WinY:RelCoord);
Schreibt «Put» ab Position «PutX» an die Koordinaten «WinX,WinY».

PROCEDURE PutClue(Clue:ARRAY OF CHAR);
Schreibt «Clue» in Zeile 25 und installiert, bzw. löscht sie.

PROCEDURE PutError(Error:ARRAY OF CHAR);
Schreibt «Error» in Zeile 24 und installiert, bzw. löscht sie.

PROCEDURE PutOnTop;
Sorgt dafür, dass das Fenster zuoberst, damit aktiv und voll sichtbar ist.

PROCEDURE SetCursor(Flag:BOOLEAN);
Schaltet den Cursor gemäss «Flag» ein und aus.

PROCEDURE SetLinkPosXY(Flag:BOOLEAN);
Die Fenster werden beim öffnen automatisch an den Vorgänger gelinkt und damit auf dem Bildschirm positioniert. Mit TRUE in «Flag» wird der Linkvorgang aktualisiert, mit FALSE ausgeklinkt.

PROCEDURE SetMode(Title:ARRAY OF CHAR; Mode:TitleMode);
Schreibt den Inhalt von «Titel» im Rahmen an die Stelle gemäss «Mode» (NoTitle,LeftUpperTitle, CenterUpperTitle,RightUpperTitle,LeftLowerTitle,CenterLowerTitle,RightLowerTitle).

PROCEDURE Setup;
Verzweigt mit dem Fensterpacket zur Farbgebung.

Line1-Fenster

```
TYPE
        prL1;

        CLASS cL1;
          L1 : prL1;
          PROCEDURE Add(Item:ARRAY OF CHAR; PickNr:SHORTCARD);
          PROCEDURE ChangePal;
          PROCEDURE ClearBeam;
          PROCEDURE Close;
          PROCEDURE GetBeamCh(VAR Buff:ARRAY OF CHAR);
          PROCEDURE Hide;
          PROCEDURE Open(Clue:ARRAY OF CHAR; VAR Pal:rPal; Tex:pcText);
          PROCEDURE Pick(VAR Kb:CHAR):SHORTCARD;
          PROCEDURE PutClue(Clue:ARRAY OF CHAR);
          PROCEDURE PutError(Error:ARRAY OF CHAR);
          PROCEDURE PutOnTop;
          PROCEDURE SetPickNr(PickNr:SHORTCARD);
          PROCEDURE Setup;
        END cL1;
```

Diese Klasse realisiert ein Auswahlfenster (ohne Rahmen) in Zeile 1. Es ist jeweils das oberste Fenster einer Anwendung. Die wichtigsten Prozeduren im Ablauf sind: **Open**, **Add**, **SetPickNr**, **Pick**(Loop), **Close**.

PROCEDURE Add(Item:ARRAY OF CHAR; PickNr:SHORTCARD);
Die «Item» werden in fortlaufender Folge installiert. Die «PickNr» 1..MAX(SHORTCARD) dient beim späteren Auswahlverfahren als Identifikation.

PROCEDURE ChangePal;
Bewirkt, dass das Fensterpacket die neue Farbpalette in «Pal» benutzt.

PROCEDURE Clear;
Entfernt alle Elemente aus dem Fenster.

PROCEDURE ClearBeam;
Löscht den Cursor-Balken und schaltet ihn aus.

PROCEDURE Close;
Schliesst das Fenster.

PROCEDURE GetBeamCh(VAR Buff:ARRAY OF CHAR);
Liefert den Inhalt des aktiven Cursor-Balkens.

PROCEDURE Hide;
Macht das Fenster unsichtbar.

PROCEDURE Open(Clue:ARRAY OF CHAR; VAR Pal:rPal; Tex:pcText);
Öffnet das Fenster in Zeile 1. In «Pal» wird die Farbpalette und in «Tex» der Pointer auf dieTextdaten übergeben.

PROCEDURE Pick(VAR Kb:CHAR):SHORTCARD;
Startet das Auswahlprozedere. Die Rückkehr erfolgt erst wenn ausgewählt, oder abgebrochen wurde. In «Kb» steht das Abbruchereignis. Als Funktionswert wird die Pick-Nr(AddItem) bzw. Null zurückgegeben.

PROCEDURE PutClue(Clue:ARRAY OF CHAR);
Schreibt «Clue» in Zeile 25 und installiert, bzw. löscht sie.

PROCEDURE PutError(Error:ARRAY OF CHAR);
Schreibt «Error» in Zeile 24 und installiert, bzw. löscht sie.

PROCEDURE PutOnTop;
Sorgt dafür, dass das Fenster zuoberst, damit aktiv und voll sichtbar ist.

PROCEDURE SetPickNr(PickNr:SHORTCARD);
Setzt den Cursor-Balken auf das Item mit «PickNr»

PROCEDURE Setup;
Verzweigt mit dem Fensterpacket zur Farbgebung.

Map-Fenster

```
TYPE
        tPal  = PaletteRange;

        tAttr = (Skip,Cap, (* accepted for Add *)
                 Bool,Char,Choice,Date,ShortCard,Card,LongCard,ShortInt,Int,LongInt,Real,LongReal,
                 changed,error,put);
        sAttr = SET OF tAttr;

        tAct  = PROCEDURE(ADDRESS,VAR CHAR);
```

Prozedur nach Keyboard-Aktionen. ADDRESS zeigt auf die zugeordneten Daten. CHAR liefert die Keyboard-funktion.

```
        tUpd  = PROCEDURE(ADDRESS,VAR CHAR,VAR BOOLEAN);
```

Prozedur nach Änderungsoperationen. ADDRESS zeigt auf die geänderten Daten. CHAR liefert die Keyboard-funktion. BOOLEAN steuert die Aktualisierung dieser Daten auf dem Bildschirm, sollten sie hier geändert worden sein.

```
        tArea = PROCEDURE(ADDRESS,CARDINAL,CHAR);
```

Prozedur über einen Bereich, in dem z.B. geblättert werden kann. ADDRESS ist die Kennung für den Bereich. CARDINAL enthält die relative Cursorzeile (1..n). CHAR liefert die Keyboardfunktion.

```
        prMap;
        CLASS cMap;
          Map : prMap;
          PROCEDURE AcceptKey;
          PROCEDURE ActionAdr(Adr:ADDRESS; Kb:CHAR);
          PROCEDURE AddArea(x1,y1,x2,y2:RelCoord;UpDn,Other:tArea):ADDRESS;
          PROCEDURE AddBool(VAR B:BOOLEAN; x,y:RelCoord; Attr:sAttr;
                                Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                                Clue:ARRAY OF CHAR);
          PROCEDURE AddCard(VAR C:CARDINAL; x,y:RelCoord;
                                Body:CARDINAL; Pict:sPict; Attr:sAttr;
                                Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                                Clue:ARRAY OF CHAR);
          PROCEDURE AddChar(VAR Chr:ARRAY OF CHAR; x,y:RelCoord;
                                Body:CARDINAL; Attr:sAttr;
                                Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                                Clue:ARRAY OF CHAR);
          PROCEDURE AddChoice(VAR Chr:ARRAY OF CHAR; x,y:RelCoord;
                                Body:CARDINAL; VAR Nr:CARDINAL; Attr:sAttr;
                                Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                                Clue:ARRAY OF CHAR);
          PROCEDURE AddDate(VAR D:rDate; x,y:RelCoord;
                                Body:ARRAY OF CHAR; Attr:sAttr;
                                Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                                Clue:ARRAY OF CHAR);
          PROCEDURE AddInt(VAR I:INTEGER; x,y:RelCoord;
                                Body:INTEGER; Pict:sPict; Attr:sAttr;
                                Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                                Clue:ARRAY OF CHAR);
          PROCEDURE AddLongCard(VAR LC:LONGCARD; x,y:RelCoord;
                                Body:CARDINAL; Pict:sPict; Attr:sAttr;
                                Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                                Clue:ARRAY OF CHAR);
          PROCEDURE AddLongInt(VAR LI:LONGINT; x,y:RelCoord;
```

```
                        Body:INTEGER; Pict:sPict; Attr:sAttr;
                        Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                        Clue:ARRAY OF CHAR);
    PROCEDURE AddLongReal(VAR LR:LONGREAL; x,y:RelCoord;
                        Body:REAL; Pict:sPict; Attr:sAttr;
                        Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                        Clue:ARRAY OF CHAR);
    PROCEDURE AddReal(VAR R:REAL; x,y:RelCoord;
                        Body:REAL; Pict:sPict; Attr:sAttr;
                        Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                        Clue:ARRAY OF CHAR);
    PROCEDURE AddShortCard(VAR SC:SHORTCARD; x,y:RelCoord;
                        Body:SHORTCARD; Pict:sPict; Attr:sAttr;
                        Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                        Clue:ARRAY OF CHAR);
    PROCEDURE AddShortInt(VAR SI:SHORTINT; x,y:RelCoord;
                        Body:SHORTINT; Pict:sPict; Attr:sAttr;
                        Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                        Clue:ARRAY OF CHAR);
    PROCEDURE AddText(Text:ARRAY OF CHAR; x,y:RelCoord;
                        Body:CARDINAL; Pos:tPos; Pal:tPal);
    PROCEDURE ChangePal;
    PROCEDURE ClearBeam;
    PROCEDURE Close;
    PROCEDURE Delay;
    PROCEDURE DeleteAdr(Adr:ADDRESS);
    PROCEDURE DeleteArea(Adr:ADDRESS);
    PROCEDURE DeleteXY(x,y:RelCoord);
    PROCEDURE ExistsAdr(Adr:ADDRESS):BOOLEAN;
    PROCEDURE ExistsChangeAdr(Adr:ADDRESS; Clear:BOOLEAN):BOOLEAN;
    PROCEDURE ExistsConfirm(Clear:BOOLEAN):BOOLEAN;
    PROCEDURE ExistsError():BOOLEAN;
    PROCEDURE Get(Return:sKb; VAR Kb:CHAR; VAR Change:BOOLEAN);
    PROCEDURE GetLink():rLink;
    PROCEDURE GotoAdr(Adr:ADDRESS);
    PROCEDURE GotoEnd;
    PROCEDURE GotoError;
    PROCEDURE GotoHome;
    PROCEDURE GotoNext;
    PROCEDURE GotoPosX(X:CARDINAL);
    PROCEDURE GotoPrev;
    PROCEDURE GotoXY(x,y:RelCoord);
    PROCEDURE Hide;
    PROCEDURE MoveCoord(X,Y:INTEGER; Ctrl,Frame:BOOLEAN);
    PROCEDURE Open(X1,Y1,X2,Y2:AbsCoord; Frame:BOOLEAN;
                        Title,Clue:ARRAY OF CHAR; VAR Pal:rPal; Tex:pcText);
    PROCEDURE PutAdr(Adr:ADDRESS);
    PROCEDURE PutClue(Clue:ARRAY OF CHAR);
    PROCEDURE PutError(Error:ARRAY OF CHAR);
    PROCEDURE PutErrorAdr(Adr:ADDRESS; Error:ARRAY OF CHAR);
    PROCEDURE PutOnTop;
    PROCEDURE PutWarnAdr(Adr:ADDRESS; Confirm:BOOLEAN;Warn:ARRAY OF CHAR);
    PROCEDURE Refresh;
    PROCEDURE RefreshAdr(Adr:ADDRESS; ChsX:CARDINAL);
    PROCEDURE SetAdr(Adr:ADDRESS; Attr:sAttr;
                        Upd:tUpd; Act:tAct; ActKey:sKb;
                        Clue:ARRAY OF CHAR);

    PROCEDURE SetArea(Adr:ADDRESS; Bottom:CARDINAL; Keys:sKb;
                        Clue:ARRAY OF CHAR);
    PROCEDURE SetChange(Flag:BOOLEAN);
    PROCEDURE SetChangeAdr(Adr:ADDRESS; Flag:BOOLEAN);
    PROCEDURE SetCharAdr(Adr:ADDRESS; ChsX:CARDINAL);
```

```
    PROCEDURE SetClueAdr(Adr:ADDRESS; Clue:ARRAY OF CHAR);
    PROCEDURE SetCtrl(Ctrl,Frame:BOOLEAN);
    PROCEDURE SetCursor(Flag:BOOLEAN);
    PROCEDURE SetErrorAdr(Adr:ADDRESS; Flag:BOOLEAN);
    PROCEDURE SetErrorReturn(Flag:BOOLEAN);
    PROCEDURE SetLink(VAR Link:rLink);
    PROCEDURE SetLinkPosXY(Flag:BOOLEAN);
    PROCEDURE SetMode(Title:ARRAY OF CHAR; Mode:TitleMode);
    PROCEDURE SetOkMsg(Key:CHAR; Msg:ARRAY OF CHAR);
    PROCEDURE SetPalAdr(Adr:ADDRESS; Pal:tPal);
    PROCEDURE SetReturn(Return:sKb);
    PROCEDURE SetSkipAdr(Adr:ADDRESS; Flag:BOOLEAN);
    PROCEDURE Setup;
    PROCEDURE UpdateAdr(Adr:ADDRESS; Kb:CHAR);
    PROCEDURE WhereMaxX():RelCoord;
    PROCEDURE WhereMaxY():RelCoord;
    PROCEDURE WhichAdrX(Adr:ADDRESS):RelCoord;
    PROCEDURE WhichAdrY(Adr:ADDRESS):RelCoord;
END cMap;
```

Diese Klasse realisiert die eigentliche Datenbearbeitung am Bildschirm. Sie behandelt den Normalfall selbständig, lässt dem Informatiker aber weiten Spielraum zur Beeinflussung des Systems. So lassen sich z.B. auf Feldebene Exits installieren welche auf Änderungen reagieren und vor verlassen des Feldes (x1,x2,y) angesprungen werden, oder solche, die auf bestimmte Ereignisse reagieren. Ebenso können ganze Bereiche mit den entprechenden Exits versehen werden. Die Klasse speichert die Daten kein zweites Mal. Sie verwaltet lediglich Adressen zu den Daten. Nach Installation (Add) der Daten in der Map lassen sie sich dort immer mit ihrer Original-Adresse ansprechen. Die Aufbereitung von Ein- und Ausgabe behandelt sie autonom. Map-Felder (Ein-/Ausgabe) sind als virtuelle Ausschnitte auf Daten beliebiger Länge realisiert. Sie können jederzeit entfernt, oder neu hinzugefügt werden. Falls z.B. die zugrundeliegenden Daten geändert haben, genügt ein einziger Befehl (Refresh), und das Bild ist aktualisiert. Die wichtigsten Prozeduren im Ablauf sind: **Open**, **Add..**, **SetOkMsg**, **Get**(Loop), **Refresh** und **Close**.

Bei der Beschreibung der einzelnen Prozeduren wird auf die Wiederholung grundsätzlicher Definitionen verzichtet. Sie sind deshalb an dieser Stelle aufgeführt:

Act:tAct; ActKey:sKb;
Prozedur «Act», welche die Ereignisse gemäss «ActKey» behandelt. NULLPROC ist erlaubt.

Attr:sAttr;
Attribut «Cap» wandelt Klein- in Grossbuchstaben um (Com_1.ChCap) und «Skip» sperrt die Eingabe. Ein Skip-Feld ohne installierte Aktionsprozedur wird vom Cursor-Balken übersprungen.

Clue:ARRAY OF CHAR;
Hinweise für Zeile 25.

Pal:tPal;
Farbindex 0..8 (vgl. Konstanten am Anfang von Scr_1: z.B. Ground0, Frame1 etc.) welche die Farbgebung bestimmen.

Pict:sPict;
Picture für die Darstellung von Zahlen. In Com_1 ist das Set beschrieben und sind auch vordefinierte Konstanten verfügbar. Massgebend für die Feldlänge zur Darstellung der Zahl (ist immer voll sichtbar) ist zusätzlich die Angabe in «Body» welche Vorzeichen sowie die Stellen vor und nach dem Dezimalpunkt steuern. Die Prozedur Com_1.SizeLR berechnet jeweils die Feldgrösse zur Darstellung.

Upd:tUpd;
Prozedur «Upd», welche nach einer Änderung, direkt vor dem verlassen des Feldes aufgerufen wird. NULLPROC ist erlaubt.

PROCEDURE AcceptKey;
Prüft auf [AltS], [AltX] und [Esc] und zeigt in Zeile 24 an, dass die gedrückte Taste akzeptiert ist, aber erst später behandelt wird.

```
PROCEDURE ActionAdr(Adr:ADDRESS; Kb:CHAR);
```

Löst mit der Datenadresse «Adr» künstlich die Aktionsprozedur aus, welche für die Funktion «Kb» installiert ist

```
PROCEDURE AddArea(x1,y1,x2,y2:RelCoord;UpDn,Other:tArea):ADDRESS;
```

Installiert einen Bereich, der alle Felder innerhalb der angegeben Koordinaten «x1,y1,x2,y2» umfasst (x dürfen ungenau sein). «UpDn» ist eine Prozedur für Blättern, «Other» für andere Ereignisse. Für beide ist auch NULLPROC zulässig. Im Funktionswert wird die Kennung für diesen Bereich zurückgegeben.

```
PROCEDURE AddBool(VAR B:BOOLEAN; x,y:RelCoord; Attr:sAttr;
                  Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                  Clue:ARRAY OF CHAR);
```

Installiert die bool'sche Variable «B» in den Koordinaten «x,y». Sie belegt fix 1 Stelle.

```
PROCEDURE AddCard(VAR C:CARDINAL; x,y:RelCoord;
                  Body:CARDINAL; Pict:sPict; Attr:sAttr;
                  Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                  Clue:ARRAY OF CHAR);
```

Installiert Cardinal «C» in den Koordinaten «x,y». «Body» bestimmt die Stellenzahl. Z.B. 2 für Zahlen von 0..99.

```
PROCEDURE AddChar(VAR Chr:ARRAY OF CHAR; x,y:RelCoord;
                  Body:CARDINAL; Attr:sAttr;
                  Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                  Clue:ARRAY OF CHAR);
```

Installiert den String «Chr» in den Koordinaten «x,y». «Body» bestimmt den sichtbaren Ausschnitt.

```
PROCEDURE AddChoice(VAR Chr:ARRAY OF CHAR; x,y:RelCoord;
                  Body:CARDINAL; VAR Nr:CARDINAL; Attr:sAttr;
                  Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                  Clue:ARRAY OF CHAR);
```

Installiert ein Auswahlfeld in den Koordinaten «x,y». «Body» bestimmt den sichtbaren Ausschnitt. Er lässt sich mit der Prozedur GetChoiceBody dynamisch ermitteln. In «Chr» wird der Item-String erwartet. Also z.B. 'Wahl0, Wahl1,,Wahl3'. In «Nr» wird die Ordnungszahl des aktiven (sichtbaren, ausgewählten) Items übergeben und ist damit das persistente Objekt (und nicht «Chr»).

```
PROCEDURE AddDate(VAR D:rDate; x,y:RelCoord;
                  Body:ARRAY OF CHAR; Attr:sAttr;
                  Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                  Clue:ARRAY OF CHAR);
```

Installiert das Datum «D» in den Koordinaten «x,y». «Body» enthält die Datummaske und bestimmt damit den sichtbaren Ausschnitt.

```
PROCEDURE AddInt(VAR I:INTEGER; x,y:RelCoord;
                  Body:INTEGER; Pict:sPict; Attr:sAttr;
                  Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                  Clue:ARRAY OF CHAR);
```

Installiert Integer «I» in den Koordinaten «x,y». «Body» bestimmt tatsächliche Stellenzahl und Vorzeichen. Z.B. -2 für Zahlen von -99..+99.

```
PROCEDURE AddLongCard(VAR LC:LONGCARD; x,y:RelCoord;
                  Body:CARDINAL; Pict:sPict; Attr:sAttr;
                  Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                  Clue:ARRAY OF CHAR);
```

Installiert LongCardinal «LC» in den Koordinaten «x,y». «Body» bestimmt tatsächliche Stellenzahl. Z.B. 6 für Zahlen vom 0..999999

```
PROCEDURE AddLongInt(VAR LI:LONGINT; x,y:RelCoord;
                  Body:INTEGER; Pict:sPict; Attr:sAttr;
                  Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                  Clue:ARRAY OF CHAR);
```

Installiert LongInteger «LI» in den Koordinaten «x,y». «Body» bestimmt tatsächliche Stellenzahl und Vorzeichen. Z.B. -6 für Zahlen von -999999..+999999.

```
PROCEDURE AddLongReal(VAR LR:LONGREAL; x,y:RelCoord;
                      Body:REAL; Pict:sPict; Attr:sAttr;
                      Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                      Clue:ARRAY OF CHAR);
```

Installiert LongReal «LR» in den Koordinaten «x,y». «Body» bestimmt tatsächliche Stellenzahl und Vorzeichen. Z.B. -6.3 für Zahlen von -999999.999..+999999.999

```
PROCEDURE AddReal(VAR R:REAL; x,y:RelCoord;
                  Body:REAL; Pict:sPict; Attr:sAttr;
                  Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                  Clue:ARRAY OF CHAR);
```

Installiert Real «R» in den Koordinaten «x,y». «Body» bestimmt tatsächliche Stellenzahl und Vorzeichen. Z.B. -3.2 für Zahlen von -999.99..+999.99

```
PROCEDURE AddShortCard(VAR SC:SHORTCARD; x,y:RelCoord;
                       Body:SHORTCARD; Pict:sPict; Attr:sAttr;
                       Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                       Clue:ARRAY OF CHAR);
```

Installiert ShortCardinal «SC» in den Koordinaten «x,y». «Body» bestimmt tatsächliche Stellenzahl . Z.B. 2 für 0..99.

```
PROCEDURE AddShortInt(VAR SI:SHORTINT; x,y:RelCoord;
                      Body:SHORTINT; Pict:sPict; Attr:sAttr;
                      Pal:tPal; Upd:tUpd; Act:tAct; ActKey:sKb;
                      Clue:ARRAY OF CHAR);
```

Installiert ShortInteger «SI» in den Koordinaten «x,y». «Body» bestimmt tatsächliche Stellenzahl und Vorzeichen. Z.B. -2 für Zahlen von -99..+99.

```
PROCEDURE AddText(Text:ARRAY OF CHAR; x,y:RelCoord;
                  Body:CARDINAL; Pos:tPos; Pal:tPal);
```

Installiert den String «Text» (z.B. Überschriften) in den Koordinaten «x,y». «Body» bestimmt die Darstellungsbreite und «Pos» die Ausrichtung von «Text» darin (Tex_1.tPos = Nothing,Left,Center,Right).

```
PROCEDURE ChangePal;
```

Bewirkt, dass das Fensterpacket die neue Farbpalette in «Pal» benutzt.

```
PROCEDURE ClearBeam;
```

Löscht den Cursor-Balken und schaltet ihn aus.

```
PROCEDURE Close;
```

Schliesst das Fenster.

```
PROCEDURE Delay;
```

Verzögert die Programmausführung und behandelt dabei [AltEnd], [AltHome], [+] und [-], welche die Verzögerung entsprechend beeinflussen. Faktor und Startoption können vom Benutzer angepasst werden.

```
PROCEDURE DeleteAdr(Adr:ADDRESS);
```

Entfernt das Feld mit der Adresse «Adr»

```
PROCEDURE DeleteArea(Adr:ADDRESS);
```

Entfernt den Bereich (nicht die Daten) mit der Kennung «Adr».

```
PROCEDURE DeleteXY(x,y:RelCoord);
```

Entfernt das Feld unter den angegebenen Koordinaten.

```
PROCEDURE ExistsAdr(Adr:ADDRESS):BOOLEAN;
```

Beantwortet im Funktionswert die Frage nach der Installation der Datenadresse «Adr» .

```
PROCEDURE ExistsChangeAdr(Adr:ADDRESS; Clear:BOOLEAN):BOOLEAN;
```

Beantwortet im Funktionswert die Frage nach Änderungen (Eingabe) auf der Datenadresse «Adr» und löscht gleichzeitig das Änderungsflag, wenn «Clear» verlangt ist.

PROCEDURE ExistsConfirm(Clear:BOOLEAN):BOOLEAN;
Beantwortet im Funktionswert die Frage nach der Existenz von Warnmeldungen mit Bestätigung (aus PutWarnAdr) und löscht sie gleichzeitig, wenn «Clear» verlangt wird.

PROCEDURE ExistsError():BOOLEAN;
Beantwortet im Funktionswert die Frage nach der Existenz von Fehlern in der Map.

PROCEDURE Get(Return:sKb; VAR Kb:CHAR; VAR Change:BOOLEAN);
Übergibt die Kontrolle der Klasse welche sie solange behält, bis ein (unbehandeltes) Ereignis eintritt, welches in «Return» aufgeführt ist. Danach enthält «Kb» das Ereignis, und «Change» den Änderungsstatus.

PROCEDURE GetLink():rLink;
Liefert im Funktionswert die Angaben über den aktuellen Link-Status.

PROCEDURE GotoAdr(Adr:ADDRESS);
Stellt den Cursor-Beam an die Datenadresse«Adr», bzw. die nächstlogische bei gesperrtem Feld.

PROCEDURE GotoEnd;
Stellt den Cursor-Beam auf das letzte, ungesperrte Feld.

PROCEDURE GotoError;
Stellt den Cursor-Beam auf das erste Fehler-Feld.

PROCEDURE GotoHome;
Stellt den Cursor-Beam auf das erste, ungesperrte Feld.

PROCEDURE GotoNext;
Stellt den Cursor-Beam auf das nächste, ungesperrte Feld.

PROCEDURE GotoPosX(X:CARDINAL);
Verschiebt den virtuellen Ausschnitt auf einem aktiven String-Feld an die Position von «X» mit 0..HIGH(String).

PROCEDURE GotoPrev;
Stellt den Cursor-Beam auf das vorangehende, ungesperrte Feld.

PROCEDURE GotoXY(x,y:RelCoord);
Stellt den Cursor-Beam auf das ungesperrte Feld unter «x,y», bzw das nächste ungesperrte.

PROCEDURE Hide;
Macht das Fenster unsichtbar.

PROCEDURE MoveCoord(X,Y:INTEGER; Ctrl,Frame:BOOLEAN);
Verschiebt die Map um den Wert von «X,Y». Z.B. -2,+3 verschieben um 2 Stellen nach links und 3 Stellen nach unten. «Ctrl» steuert die Anzeige des Rundlaufverhaltens und «Frame» den Rahmentypen (= für TRUE, - für FALSE).

PROCEDURE Open(X1,Y1,X2,Y2:AbsCoord; Frame:BOOLEAN;
Title,Clue:ARRAY OF CHAR; VAR Pal:rPal; Tex:pcText);
Öffnet die Map in den angegebenen Koordinaten. «Frame» steuert die Verwendung des Rahmens. In «Pal» wird die Farbpalette und in «Tex» der Pointer auf Textdaten übergeben. «Titel» wird in die Kopfmitte des Rahmens (sofern vorhanden), und «Clue» in Zeile 25 geschrieben.

PROCEDURE PutAdr(Adr:ADDRESS);
Schreibt den Inhalt der Daten von «Adr» neu auf den Bildschirm, ohne Rahmenbedingungen zu verändern. Dies im Unterschied zu Refresh und RefreshAdr.

PROCEDURE PutClue(Clue:ARRAY OF CHAR);
Installiert, bzw. löscht «Clue» auf Map-Ebene in Zeile 25, ist aber nur sichtbar, wenn das Feld, auf dem der Cursor positioniert ist, keine eigene besitzt.

PROCEDURE PutError(Error:ARRAY OF CHAR);
Installiert bzw. löscht «Error» auf Map-Ebene in Zeile 24, ist aber nur sichtbar, wenn das Feld, auf dem der Cursor positioniert ist, keine eigene besitzt.

PROCEDURE PutErrorAdr(Adr:ADDRESS; Error:ARRAY OF CHAR);
Installiert «Error» an die Datenadresse «Adr» und setzt dort das Fehlermerkmal. Beide werden zum logisch richtigen Zeipunkt automatisch gelöscht.

PROCEDURE PutOnTop;
Sorgt dafür, dass das Fenster zuoberst, damit aktiv und voll sichtbar ist.

PROCEDURE PutWarnAdr(Adr:ADDRESS; Confirm:BOOLEAN;Warn:ARRAY OF CHAR);
Installiert «Warn» und «Confirm» an die Datenadresse «Adr» . Wird «Confirm» bejaht, muss die Meldung zwingend bestätigt werden. Beide werden zum logisch richtigen Zeipunkt automatisch gelöscht.

PROCEDURE Refresh;
Aktualisiert die ganze Map und ändert deren Rahmenbedingungen auf Neustart (alle Sünden sind vergeben!). Dies ist notwendig, wenn die Datenbasis der Map, oder wesentlicher Teile davon, geändert hat (z.B. neuer Datenstrom).

PROCEDURE RefreshAdr(Adr:ADDRESS; ChsX:CARDINAL);
Aktualisiert, im Unterschied zu Refresh, nur die Datenadresse«Adr» .

PROCEDURE SetAdr(Adr:ADDRESS; Attr:sAttr;
Upd:tUpd; Act:tAct; ActKey:sKb;
Clue:ARRAY OF CHAR);
Ändert die Installation der Datenadresse«Adr» .

PROCEDURE SetArea(Adr:ADDRESS; Bottom:CARDINAL; Keys:sKb;
Clue:ARRAY OF CHAR);
Aktualisiert den Bereich mit Kennung «Adr». «Bottom» steht für letzte belegte Zeile des Bereichs (relativ, 0..MAX(Bereich)) und «Keys» für Ereignisse mit Exit-Wirkung (AddArea). Dabei werden [PgDn], [PgUp], [ArrDn] und [ArrUp] automatisch der Prozedur «UpDn» zugewiesen, und nur die restlichen der Prozedur «Other»

PROCEDURE SetChange(Flag:BOOLEAN);
Setzt das globale Änderungsmerkmal in den Zustand von «Flag»

PROCEDURE SetChangeAdr(Adr:ADDRESS; Flag:BOOLEAN);
Setzt das Änderungsmerkmal für die Datenadresse «Adr» in den Zustand von «Flag»

PROCEDURE SetCharAdr(Adr:ADDRESS; ChsX:CARDINAL);
Verschiebt den virtuellen Ausschnitt auf der Datenadresse «Adr» an die Position von «ChsX» mit 0..HIGH(String).

PROCEDURE SetClueAdr(Adr:ADDRESS; Clue:ARRAY OF CHAR);
Installiert, bzw. löscht «Clue» an der Datenadresse «Adr», ist aber nur sichtbar, wenn der Cursor auf dem Feld positioniert ist.

PROCEDURE SetCtrl(Ctrl,Frame:BOOLEAN);
Setzt «Ctrl» für die Anzeige des Rundlaufverhaltens und «Frame» für den Rahmentypen (= für TRUE, - für FALSE).

PROCEDURE SetCursor(Flag:BOOLEAN);
Schaltet den Cursor gemäss «Flag» ein und aus.

PROCEDURE SetErrorAdr(Adr:ADDRESS; Flag:BOOLEAN);
Schaltet das Fehlermerkmal für die Datenadresse «Adr» gemäss «Flag» ein und aus.

PROCEDURE SetErrorReturn(Flag:BOOLEAN);
Schaltet das Merkmal, das bei Fehlern die Prozedur Get zur Rückgabe der Kontrolle veranlasst, gemäss «Flag» ein und aus (aus ist Standard).

PROCEDURE SetLink(VAR Link:rLink);
Installiert «Link» (Anbindung an einen Vorgänger).

PROCEDURE SetLinkPosXY(Flag:BOOLEAN);
Die Fenster werden beim öffnen automatisch an den Vorgänger gelinkt und damit auf dem Bildschirm positioniert. Mit TRUE in «Flag» wird der Linkvorgang aktualisiert, mit FALSE ausgeklinkt.

PROCEDURE SetMode(Title:ARRAY OF CHAR; Mode:TitleMode);
Schreibt den Inhalt von «Titel» im Rahmen an die Stelle gemäss «Mode» (NoTitle,LeftUpperTitle, CenterUpperTitle,RightUpperTitle,LeftLowerTitle,CenterLowerTitle,RightLowerTitle).

PROCEDURE SetOkMsg(Key:CHAR; Msg:ARRAY OF CHAR);
Installiert «Msg» und «Key» als akzeptierten Auslöser einer Aktion auf Map-Ebene (z.B. Freigabe zur Nachführung einer DB). Sie werden automatisch aktiviert und deaktiviert, abhängig davon, ob «Key» in Return (Get, SetReturn) enthalten ist, oder nicht.

PROCEDURE SetPalAdr(Adr:ADDRESS; Pal:tPal);
Installiert die Farbpalette «Pal» an der Datenadresse «Adr».

PROCEDURE SetReturn(Return:sKb);
Installiert mit «Return» neue Rückkehrbedingungen für die Prozedur Get.

PROCEDURE SetSkipAdr(Adr:ADDRESS; Flag:BOOLEAN);
Schaltet das Skip-Attribut an der Datenadresse «Adr» gemäss «Flag».

PROCEDURE Setup;
Verzweigt mit dem Fensterpacket zur Farbgebung.

PROCEDURE UpdateAdr(Adr:ADDRESS; Kb:CHAR);
Löst mit der Datenadresse «Adr» künstlich die Änderungsprozedur aus.

PROCEDURE WhereMaxX():RelCoord;
Liefert im Funktionswert den Höchstwert von x. (für Fenster mit Rahmen: 1..78).

PROCEDURE WhereMaxY():RelCoord;
Liefert im Funktionswert den Höchstwert von y. (für Fenster mit Rahmen: 1..20).

PROCEDURE WhichAdrX(Adr:ADDRESS):RelCoord;
Liefert im Funktionswert die x-Koordinate, an welcher die Datenadresse «Adr» installiert ist.

PROCEDURE WhichAdrY(Adr:ADDRESS):RelCoord;
Liefert im Funktionswert die y-Koordinate, an welcher die Datenadresse «Adr» installiert ist.

Pop-up-Fenster

```
TYPE
        prPop;

        CLASS cPop;
          Pop : prPop;
          PROCEDURE Add(Item:ARRAY OF CHAR; PickNr:SHORTCARD);
          PROCEDURE ChangePal;
          PROCEDURE ClearBeam;
          PROCEDURE Close;
          PROCEDURE GetBeamCh(VAR Buff:ARRAY OF CHAR);
          PROCEDURE Hide;
          PROCEDURE Open(Title,Clue:ARRAY OF CHAR; VAR Pal:rPal; Tex:pcText);
          PROCEDURE Pick(VAR Kb:CHAR):SHORTCARD;
          PROCEDURE PutClue(Clue:ARRAY OF CHAR);
          PROCEDURE PutError(Error:ARRAY OF CHAR);
          PROCEDURE PutOnTop;
          PROCEDURE SetLinkPosXY(Flag:BOOLEAN);
          PROCEDURE SetPickNr(PickNr:SHORTCARD);
          PROCEDURE Setup;
        END cPop;
```

Diese Klasse realisiert ein Pop-up-Fenster. Die wichtigsten Prozeduren im Ablauf sind: **Open**, **Add**, **SetPickNr**, **Pick**(Loop), **Close**.

PROCEDURE Add(Item:ARRAY OF CHAR; PickNr:SHORTCARD);
Die «Item» werden in fortlaufender Folge installiert. Die «PickNr» 1..MAX(SHORTCARD) dient beim späteren Auswahlverfahren als Identifikation.

PROCEDURE ChangePal;
Bewirkt, dass das Fensterpacket die neue Farbpalette in «Pal» benutzt.

PROCEDURE ClearBeam;
Löscht den Cursor-Balken und schaltet ihn aus.

PROCEDURE Close;
Schliesst das Fenster.

PROCEDURE GetBeamCh(VAR Buff:ARRAY OF CHAR);
Liefert den Inhalt des aktiven Cursor-Balkens.

PROCEDURE Hide;
Macht das Fenster unsichtbar.

PROCEDURE Open(Title,Clue:ARRAY OF CHAR; VAR Pal:rPal; Tex:pcText);
Öffnet das Fenster. In «Pal» wird die Farbpalette und in «Tex» der Pointer auf Textdaten übergeben.

PROCEDURE Pick(VAR Kb:CHAR):SHORTCARD;
Startet das Auswahlprozedere. Die Rückkehr erfolgt erst wenn ausgewählt, oder abgebrochen wurde. In «Kb» steht das Abbruchereignis. Als Funktionswert wird die Pick-Nr (AddItem) bzw. Null zurückgegeben.

PROCEDURE PutClue(Clue:ARRAY OF CHAR);
Schreibt «Clue» in Zeile 25 und installiert, bzw. löscht sie.

PROCEDURE PutError(Error:ARRAY OF CHAR);
Schreibt «Error» in Zeile 24 und installiert, bzw. löscht sie.

PROCEDURE PutOnTop;
Sorgt dafür, dass das Fenster zuoberst, damit aktiv und voll sichtbar ist.

PROCEDURE SetLinkPosXY(Flag:BOOLEAN);
Die Fenster werden beim öffnen automatisch an den Vorgänger gelinkt und damit auf dem Bildschirm positioniert. Mit TRUE in «Flag» wird der Linkvorgang aktualisiert, mit FALSE ausgeklinkt.

PROCEDURE SetPickNr(PickNr:SHORTCARD);
Setzt den Cursor-Balken auf das Item mit «PickNr»

PROCEDURE Setup;
Verzweigt mit dem Fensterpacket zur Farbgebung.

Timer

```
TYPE
    prTimer;

    CLASS cTimer;
      Timer : prTimer;
      PROCEDURE Ask;
      PROCEDURE Close;
      PROCEDURE Hide;
      PROCEDURE Open(Pause:CARDINAL; VAR Pal:rPal; L24:ARRAY OF CHAR);
      PROCEDURE PutOnTop;
    END cTimer;
```

Diese Klasse realisiert einen Timer, welcher (a) in Zeile 24 anzeigt, dass das System (ohne Loop) arbeitet und (b) auf [Esc], [AltX] sowie [AltS] reagiert. Die wichtigsten Prozeduren im Ablauf sind: **Open**, **Ask**, **Close**.

PROCEDURE Ask;
Konsultiert den Ticker bezügliche Arbeitsfortschritt und behandelt pendente Meldungen.

PROCEDURE Close;
Schliesst den Timer.

PROCEDURE Hide;
Macht das Fenster unsichtbar.

PROCEDURE Open(Pause:CARDINAL; VAR Pal:rPal; L24:ARRAY OF CHAR);
Öffnet das Fenster auf Zeile 24. In «Pal» wird die Farbpalette und in «L24» der Text für die Anzeige übergeben.

PROCEDURE PutOnTop;
Sorgt dafür, dass das Fenster zuoberst, damit aktiv und voll sichtbar ist.

Scr_2

Einleitung

Dieser Modul realisiert Funktionen zur Benutzerorientierung des Gesamtsystems. Dies betrifft globale Farbpaletten, Bool, Datum, Bremse (Delay), Flag, Sprache, Maus, Ton, Warten (Timer) und Editor. Die Daten dazu sind in der Datei User.DAT (FIO_1) abgelegt. Die Farbpaletten für die einzelne Anwendung sind zusammen mit weiteren Steuerungsdaten in einer Datei Modul.DAT (Modul.EXE) abgelegt.

Beschreibung

```
FROM Scr_1 IMPORT cL1, rPal;
FROM Tex_1 IMPORT pcText;
FROM Window IMPORT WinType;
```

PROCEDURE Setup(Tex:pcText; VAR Pal:ARRAY OF rPal; VAR L1:cL1);
Einstiegspunkt zum Setup der Benutzerorientierung.

PROCEDURE SetupBasic(Tex:pcText; Base,L24,L25:WinType; VAR Pal:rPal);
Einstiegspunkt zur Farbgebung für ein Basis-Fenster «Base,L24,L25» . «Tex» ist der Pointer auf die Textdaten und «Pal» die Farbpalette für das Packet.

PROCEDURE SetupL1(Tex:pcText; Base,L24,L25:WinType; VAR Pal:rPal);
Einstiegspunkt zur Farbgebung für ein Zeile1-Fenster «Base,L24,L25» . «Tex» ist der Pointer auf die Textdaten und «Pal» die Farbpalette für das Packet.

PROCEDURE SetupMap(Tex:pcText; Base,L24,L25:WinType; VAR Pal:rPal);
Einstiegspunkt zur Farbgebung für ein Map-Fenster «Base,L24,L25» . «Tex» ist der Pointer auf die Textdaten und «Pal» die Farbpalette für das Packet.

PROCEDURE SetupPop(Tex:pcText; Base,L24,L25:WinType; VAR Pal:rPal);
Einstiegspunkt zur Farbgebung für ein Pop-up-Fenster «Base,L24,L25» . «Tex» ist der Pointer auf die Textdaten und «Pal» die Farbpalette für das Packet.

Scr_3

Einleitung

Dieser Modul realisiert eine Klasse zur Auswahl von Daten am Bildschirm. Die Daten werden von der Klasse verwaltet. Es stehen zwei Auswahlverfahren zur Verfügung: (a) «Pick», wenn genau 1 Element ausgewählt werden darf und (b) «Flag» für eine beliebige Anzahl Elemente.

Beschreibung

```
FROM Scr_1 IMPORT cArray, rPal;
FROM Tab_1 IMPORT cTab1, cEle1, pcEle1, tApply;
FROM Tex_1 IMPORT pcText;

TYPE
        tCap = PROCEDURE(VAR ARRAY OF CHAR); (* for Keyboardentry *)
```

PROCEDURE Locate(Source,Pattern:ARRAY OF CHAR; Cap:tCap):INTEGER;
Diese Prozedur ist speziell für cPickEle.Locate geschrieben und vergleicht «Source» und «Pattern» in der Länge von letzterem. In «Cap» wird eine Prozedur (Com_1.ChsCap, Com_1.ChsKey, NULLPROC) zur Umwandlung von Klein- in Grossbuchstaben auf «Source» erwartet. Als Funktionswert wird -1, 0, +1 (für kleiner, gleich grösser) zurückerwartet.

```
TYPE
        pcPickEle = POINTER TO cPickEle;

        CLASS cPickEle(cEle1);
           flag : BOOLEAN;
           VIRTUAL PROCEDURE AcceptFlag(p:pcEle1; VAR F7:BOOLEAN):BOOLEAN;
           VIRTUAL PROCEDURE AcceptPick(p:pcEle1):BOOLEAN;
           VIRTUAL PROCEDURE Apply(p:pcEle1);
           VIRTUAL PROCEDURE Compare(p:pcEle1):INTEGER;
           VIRTUAL PROCEDURE Make(p:pcEle1; VAR xChar:ARRAY OF CHAR);
           VIRTUAL PROCEDURE Locate(p:pcEle1; VAR Kb:ARRAY OF CHAR;Cap:tCap):BOOLEAN;
        END cPickEle;
```

Bildet die Schnittstelle zu cPick für alle ihre Elemente. Die virtuellen Prozeduren müssen, mit Ausnahme von AcceptFlag und AcceptPick, überschrieben werden.

VIRTUAL PROCEDURE AcceptFlag(p:pcEle1; VAR F7:BOOLEAN):BOOLEAN;
Diese Prozedur braucht nur überschrieben zu werden, wenn das Funktionsergebnis nicht immer TRUE ist. Sie wird aufgerufen, wenn das Element «p» auf dem Bildschirm ein Flag erhalten soll. Im Funktionswert wird die Aktion bestätigt. In «F7» kann die Funktionstaste, welche die Verarbeitung auslöst, (negativ) beeinflusst werden. Standardmässig ist «F7» aktiviert, solange mindestens 1 Element ausgewählt ist.

VIRTUAL PROCEDURE AcceptPick(p:pcEle1):BOOLEAN;
Diese Prozedur braucht nur überschrieben zu werden, wenn das Funktionsergebnis nicht immer TRUE ist. Sie wird aufgerufen, wenn das Element «p» ausgewählt wird. Im Funktionswert wird die Aktion bestätigt.

VIRTUAL PROCEDURE Apply(p:pcEle1);
In dieser Prozedur wird jedes/das ausgewählte Element «p» verarbeitet.

VIRTUAL PROCEDURE Compare(p:pcEle1):INTEGER;
Dient der Einordnung der Elemente in die interne Tabelle und liefert als Funktionswert das Ergebnis aus dem Key-Vergleich von «cPickEle» und «p». Für aufsteigend sortierte Elemente gilt: -1, 0, +1 (für kleiner, gleich, grösser).

VIRTUAL PROCEDURE Make(p:pcEle1; VAR xChar:ARRAY OF CHAR);
Die Prozedur wird mit jedem Element «p» für die Bilschirmdarstellung aufgerufen. In «xChar» wird der Bildausschnitt zurücherwartet.

VIRTUAL PROCEDURE Locate(p:pcEle1; VAR Kb:ARRAY OF CHAR;Cap:tCap):BOOLEAN;
Dient der Suche nach einem Element, welches mit der aktuellen (unsichtbaren) Eingabe am Bilschirm übereinstimmt

(auch fragmentarisch). In «Kb» wird der Keyboardbuffer übergeben (ein Leerschlag löscht den Keyboardbuffer!). «Cap» liefert die installierte Prozedur zu Umwandlung von Kleinschrift (Com_1.ChsCap, Com_1.ChsKey, NULLPROC). Im Funktionswert wird der Sachverhalt bestätigt. Bei positivem Ausgang wird der Bildausschnitt derart verschoben, dass das Element oben links steht.

♣ RETURN(Str.Pos(pcEntryEle(p)^.Entry.Name,Kb) = 0);
♣ RETURN(Scr_3.Locate(tIndex(pcIndexEle(p)^.Text^),Kb,Cap) = 0);

```
TYPE
        tAction = (page_up,page_down,array_up,array_down);
        sAction = SET OF tAction;
        rState = RECORD
           Full : BOOLEAN;
           Up   : BOOLEAN;
           Dn   : BOOLEAN;
         END;
         prPick;

        CLASS cPick(cTab1,cArray);
           PD : prPick;
           PROCEDURE Dispose;
           PROCEDURE Flag(Tex:pcText; Title:ARRAY OF CHAR; VAR Pal:rPal):BOOLEAN;
           PROCEDURE GetPos(VAR Pos:CARDINAL):pcEle1;
           PROCEDURE Init(VAR xChar:ARRAY OF CHAR; yNr:CARDINAL;Cap:tCap; Dispo:tApply);
           PROCEDURE Insert(VAR E:cPickEle);
           PROCEDURE Locate(VAR Kb:ARRAY OF CHAR; VAR Pos:CARDINAL):pcEle1;
           PROCEDURE Next(VAR Action:sAction; VAR State:rState):pcEle1;
           PROCEDURE Pick(Tex:pcText; Title:ARRAY OF CHAR; VAR Pal:rPal):BOOLEAN;
           PROCEDURE SetPos(VAR Action:sAction; VAR Pos:CARDINAL);
        END cPick;
```

Steuert den Ablauf im Kontext mit der Auswahl von Daten am Bildschirm. Die wichtigsten Prozeduren im Ablauf sind: **Init**, **Insert**, **Flag**(Loop) oder **Pick**(Loop), **Dispose**.

PROCEDURE Dispose;
Gibt die Ressourcen frei.

PROCEDURE Flag(Tex:pcText; Title:ARRAY OF CHAR; VAR Pal:rPal):BOOLEAN;
Steuert das Auswahlverfahren, wenn mehrere Elemente auszuwählen sind. In «Tex» wird der Pointer auf die Textdaten, in «Title» die Bildüberschrift, und in «Pal» die Farbpalette für das Auswahlbild erwartet. Als Funktionswert wird eine Auswahl bestätigt.

PROCEDURE GetPos(VAR Pos:CARDINAL):pcEle1;
Interne Prozedur mit Zugriff auf cPickEle.

PROCEDURE Init(VAR xChar:ARRAY OF CHAR; yNr:CARDINAL;Cap:tCap; Dispo:tApply);
Initialisiert die Klasse. «xChar» entspricht dem Bildausschnitt für 1 Element und «yNr» der Zeilenzahl. Die Spaltenzahl wird errechnet. In «Cap» wird eine Prozedur (Com_1.ChsCap, Com_1.ChsKey, NULLPROC) zur Umwandlung von Kleinschrift in «Source» erwartet, in «Dispo» eine für die Freigabe von Ressourcen der Anwendung.

PROCEDURE Insert(VAR E:cPickEle);
Installiert ein Element «E» zur Auswahl.

PROCEDURE Locate(VAR Kb:ARRAY OF CHAR; VAR Pos:CARDINAL):pcEle1;
Interne Prozedur mit Zugriff auf cPickEle.

PROCEDURE Next(VAR Action:sAction; VAR State:rState):pcEle1;
Interne Prozedur mit Zugriff auf cPickEle.

PROCEDURE Pick(Tex:pcText; Title:ARRAY OF CHAR; VAR Pal:rPal):BOOLEAN;
Steuert das Auswahlverfahren, wenn 1 Element auszuwählen ist. In «Tex» wird der Pointer auf die Textdaten, in «Title» die Bildüberschrift, und in «Pal» die Farbpalette für das Auswahlbild erwartet. Als Funktionswert wird die

Auswahl bestätigt.

PROCEDURE SetPos(VAR Action:sAction; VAR Pos:CARDINAL);
Interne Prozedur mit Zugriff auf cPickEle.

Scr_4

Einleitung

Dieser Modul realisiert die Kontrolle über alle geöffneten Fenster, ob diese sichtbar sind oder nicht.

Beschreibung

FROM Window IMPORT WinType;

PROCEDURE Close(W:WinType);
Schliesst das Fenster «W».

PROCEDURE CloseAll;
Schliesst alle Fenster.

PROCEDURE Empty():BOOLEAN;
Beantwortet im Funktionswert die Frage nach leerem Bildschirm (ohne Fenster).

PROCEDURE Exists(W:WinType):BOOLEAN;
Beantwortet im Funktionswert die Frage nach der Existenz des Fensters «W».

PROCEDURE Hide(W:WinType);
Macht das Fenster «W» unsichtbar.

PROCEDURE HideAll;
Macht alle Fenster unsichtbar. Nach PutOnTopAll ist der Ausgangzustand wieder erstellt.

PROCEDURE PutOnTop(W:WinType);
Sorgt dafür, dass das Fenster «W» zuoberst, damit aktiv und voll sichtbar ist.

PROCEDURE PutOnTopAll;
Erstellt den Ausgangszustand von HideAll.

Seq_1

Einleitung

Dieser Modul realisiert einen Automaten für Sequenzabstraktionen. Dabei können die zugrundeliegenden Datenströme aus beliebigen Quellen (Tabelle, Datei, ...) stammen. Für jeden gelesenen Datenstrom (Satz, Record, ...) kann der Anwender-Modul entscheiden ob verarbeitet, oder überlesen werden soll. Die Sequenzen können pro Gruppe auf- oder absteigend definiert sein. Die Prüfung auf deren Einhaltung lassen sich für jeden Datenstrom individuell festlegen. Bei Gruppenwechseln erhält der Benutzermodul jeweils die Kontrolle über den Abschluss der alten, und danach die Eröffnung der neuen Gruppe (erster und letzter Zyklus situativ). Er hat zudem die Möglichkeit, die ganze Verarbeitung jederzeit abzubrechen.

Beschreibung

TYPE

tPrio = SHORTCARD[1..MAX(SHORTCARD)];

Daten- und Gruppenpriorität (1 = höchste). Datenströme mit identischen Gruppenkeys werden von 1..MAX(tPrio) verarbeitet. Die Gruppen werden bei der Eröffnung von 1..MAX(tPrio), und beim Abschluss von MAX(tPrio)..1 durchlaufen.

tBreak = PROCEDURE(ADDRESS);

Prozedur für Abschluss und Eröffnung einer Gruppe. Sie erhält jeweils die Adresse auf den entsprechenden Key.

tComp = PROCEDURE(ADDRESS,ADDRESS):INTEGER;

Prozedur für den Sequenzvergleich einer Gruppe. Adresse1 und Adresse2 zeigen auf die entsprechenden Keys. Als Funktionswert wird das Ergebnis aus dem Vergleich von Key1 mit Key2 zurückerwartet, also -1, 0, +1 (für kleiner, gleich, grösser).

tMake = PROCEDURE():ADDRESS;

Prozedur, welche den Key für eine DatenGruppe erstellt. Als Funktionswert wird die Adresse des Keys zurückerwartet. Die Variable für den Key ist danach wieder frei verfügbar.

tOpen = PROCEDURE():BOOLEAN;

Prozedur, welche einen Datenstrom öffnet. Als Funktionswert wird das Ergebnis der Aktion zurückerwartet (FALSE = EOF).

tRead = PROCEDURE(VAR BOOLEAN):BOOLEAN;

Prozedur, welche einen Datenstrom bereitstellt. In der Variablen wird das Überlesen (TRUE) gesteuert. Als Funktionswert wird das Ergebnis der Aktion zurückerwartet (FALSE = EOF).

tWork = PROCEDURE();

Prozedur, welche einen Datenstrom verarbeitet.

prSequ;

```
CLASS cSequence;
   SD : prSequ;
   PROCEDURE AddData(DP:tPrio; Open:tOpen; Read:tRead; Work:tWork; Test:BOOLEAN);
   PROCEDURE AddDataGroup(DP,GP:tPrio; MakeKey:tMake);
   PROCEDURE AddGroup(GP:tPrio; Comp:tComp; KeySize:CARDINAL;
                                 Sequ:CHAR; Begin,End:tBreak);
   PROCEDURE Apply():BOOLEAN;
   PROCEDURE Cancel;
   PROCEDURE Dispose;
   PROCEDURE GetCount(DP:tPrio; VAR Rd,Wk:LONGCARD):BOOLEAN;
END cSequence;
```

AddData muss vor AddDataGroup installiert werden. Die Integrität des Systems ist spätestens bei der Prozedur Apply sichergestellt. Die Prozeduren im Ablauf sind: **AddData**, **AddDataGroup**, **AddGroup**, **Apply**(Loop[**Cancel**]), **GetCount**, **Dispose**.

PROCEDURE AddData(DP:tPrio; Open:tOpen; Read:tRead; Work:tWork; Test:BOOLEAN);
Installiert die Prozeduren «Open», «Read» und «Work» für den Datenstrom «DP». Mit «Test» wird die Prüfung auf Einhaltung der Datensequenz gesteuert.

PROCEDURE AddDataGroup(DP,GP:tPrio; MakeKey:tMake);
Installiert die Prozedur «MakeKey» des Datenstroms «DP» in Gruppe «GP» .

PROCEDURE AddGroup(GP:tPrio; Comp:tComp; KeySize:CARDINAL;
Sequ:CHAR; Begin,End:tBreak);
Installiert die Prozeduren «Comp» für den Vergleich der Keys von Gruppe «GP», «Begin» und «End» für Gruppenbeginn und -end. «KeySize» definiert den Speicherplatz, den der Gruppenkey belegt. In «Sequ» wird mit '<' oder '>' die erwartete Sequenz definiert.

PROCEDURE Apply():BOOLEAN;
Startet den Automaten. Die Rückkehr erfolgt erst, wenn alle Datenströme und Gruppen verarbeitet sind, bzw. in einer Exit-Prozedur Cancel aufgerufen wurde. Im Funktionswert wird bestätigt, dass mindestens ein Datenstrom eingelesen wurde (ohne Überlesene!).

PROCEDURE Cancel;
Bricht die Verarbeitung ab.

PROCEDURE Dispose;
Gibt die Ressourcen der Klasse frei.

PROCEDURE GetCount(DP:tPrio; VAR Rd,Wk:LONGCARD):BOOLEAN;
Liefert die Zähler «Rd» und «Wk» des Datenstroms «DP». Sie enthalten die Anzahl gelesene (Rd) und verarbeitete (Wk) Datenströme. Im Funktionswert wird die Aktion bestätigt.

SPF_1

Einleitung

Dieser Modul realisiert die Verbindung zu einem ASCII-Editor (z.B. SPF/PC von Command Technology Corporation) und der dazu notwendigen Parameterpflege.

Beschreibung

```
FROM Scr_1 IMPORT rPal, cL1;
FROM Tex_1 IMPORT pcText;
```

PROCEDURE Exec(Prg,Par,File,Word:ARRAY OF CHAR):BOOLEAN;
Erstellt aus «Par», «File» und «Word» den verlangten Übergabeparameter und ruft damit Programm «Prg» auf. Die Aktion wird im Funktionswert bestätigt. Für «Par» stehen die Schlüsselwörter «@FILE» und «@WORD» zur Verfügung. Sie werden durch den Inhalt von «File» und «Word» substituiert.

PROCEDURE Setup(VAR Prg,Par:ARRAY OF CHAR; Tex:pcText; VAR Pal:rPal);
Mit dieser Prozedur werden «Prg» (Editor-Programm) und «Par» (Parameter) festgelegt. In «Tex» wird der Pointer auf die Textdaten, und in «Pal» eine Farbpalette übergeben.

SQL_1

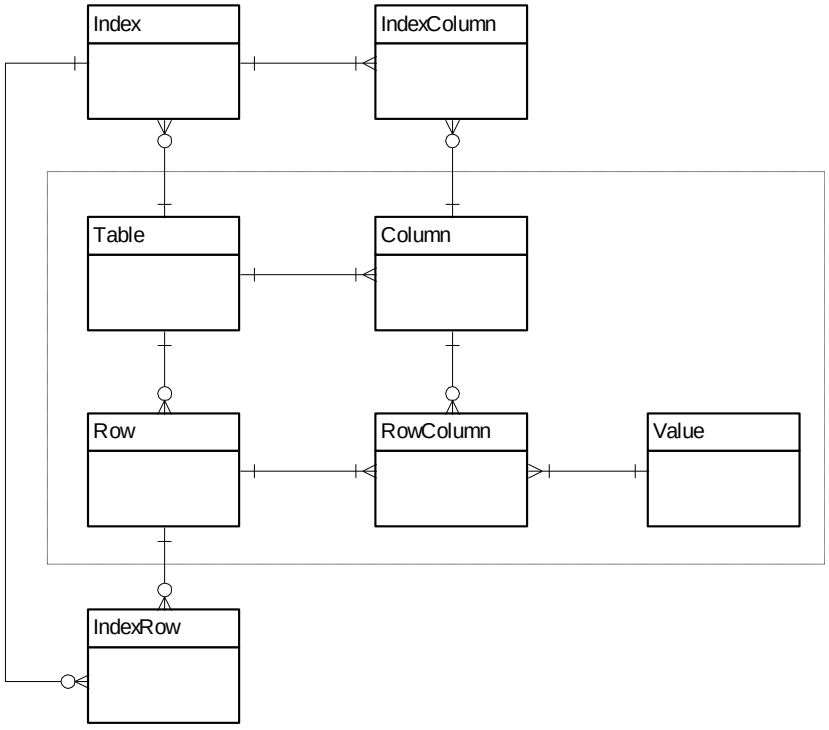
Einleitung

Dieser Modul realisiert eine relationale Datenbank samt Funktionen zu deren Pflege und Abfrage. SQL (Structured Query Language) ist die Sprache dafür. Sie wurde Mitte ‘1970 bei IBM entwickelt und hat sich seither als Standard für Mainframe und Minicomputer etabliert. Dieser Modul behandelt sowohl die interaktive, wie auch die eingebettete Form. Er optimiert die Datenablage derart, dass ein bestimmter Wert nur genau einmal physisch gespeichert wird.
.

Beschreibung

```
FIO_1 IMPORT cPage, tPath;
FROM FIO_1 IMPORT tPath;
FROM FIO_3 IMPORT tPgLn;
FROM Tab_2 IMPORT cTab2;
FROM Tex_1 IMPORT pcText, tText, tPos;
FROM Scr_1 IMPORT prPal;
TYPE
        pcSQL = POINTER TO cSQL;

        CLASS cSQL(cTab2);
            Code : INTEGER;
            Count : LONGCARD;
            PROCEDURE CloseDB;
            PROCEDURE CreateResult;
            PROCEDURE Error():tText;
            PROCEDURE ExistsDB(FP:ARRAY OF CHAR):BOOLEAN;
            PROCEDURE Init(Tex:pcText; Pal:prPal);
            PROCEDURE OpenDB(FP:ARRAY OF CHAR; RdOnly,Shared:BOOLEAN);
            PROCEDURE Query(VAR Cmd:ARRAY OF CHAR):CARDINAL;
            PROCEDURE ResultToFile(VAR TabLn,ColLn:tPgLn):tPath;
        END cSQL;
```



Tab_1

Einleitung

Dieser Modul realisiert eine Tabelle, deren interne Struktur ein ausgeglichener Baum (AVL balanced Tree) ist. Die Details können Niklaus Wirth's Buch «Algorithmen und Datenstrukturen mit Modula-2» entnommen werden.

Für jene Fälle, wo nach dem Unterbruch einer Tabellen-Operation wieder am alten Ort aufgesetzt werden soll, ist Tab_2 speziell dafür realisiert.

Der Begriff «Key» steht jeweils als Synonym für die realen Schlüsseldaten in cEle1.

Beschreibung

```
TYPE
        tBalance = SHORTINT [-1..1];
        pcEle1 = POINTER TO cEle1;

        CLASS cEle1;
           Right : pcEle1;          (* Pointer to right sub-tree *)
           Left  : pcEle1;          (* Pointer to left sub-tree  *)
           Bal   : tBalance;        (* Tree balance flag  *)
           VIRTUAL PROCEDURE Compare(p:pcEle1):INTEGER;
        END cEle1;
```

Bildet die Datenschnittstelle zu cTab1 für alle ihre Elemente die sie verwaltet. Die virtuelle Prozedur Compare muss überschrieben werden. Nur der Instanz welche cEle1 implementiert, sind ihre Daten bekannt.

```
        VIRTUAL PROCEDURE Compare(p:pcEle1):INTEGER;
```

Liefert als Funktionswert das Ergebnis aus dem Key-Vergleich von «cEle1» und «p». Für aufsteigend sortierte Elemente gilt: -1, 0, +1 (für kleiner, gleich, grösser).

```
TYPE
        tApply  = PROCEDURE(pcEle1);
```

Prozedur, welche die Daten von «pcEle1» verarbeitet.

```
        tQuality  = ARRAY[0..1]OF CHAR;
```

Definiert die Qualität des gesuchten, im Verhältnis zum angegebenen Key. Gültig sind: <, <=, =, >=, > (zweiwertige Symbole dürfen vertauscht werden).

```
        pcTab1 = POINTER TO cTab1;

        CLASS cTab1;
           Root : pcEle1;
           Curs : pcEle1;           (* cursor *)
           PROCEDURE Apply(A:tApply);
           PROCEDURE Delete(VAR E:cEle1);
           PROCEDURE Dispose;
           PROCEDURE Empty():BOOLEAN;
           PROCEDURE Find(VAR E:cEle1; Q:tQuality):BOOLEAN;
           PROCEDURE First(VAR E:cEle1):BOOLEAN;
           PROCEDURE Init;
           PROCEDURE Insert(VAR E:cEle1);
           PROCEDURE Last(VAR E:cEle1):BOOLEAN;
           PROCEDURE Replace(VAR E:cEle1);
        END cTab1;
```

Die Ordnung der Elemente in cTab1 wird von cEle1.Compare determiniert. Diese Tatsache wird bei den einzelnen Prozeduren nicht mehr erwähnt, kann jedoch von Bedeutung sein, wenn z.B. durch Umkehrung des Compare.Funktionswertes absteigend sortiert wird.

PROCEDURE Apply(A:tApply);
Ruft mit jedem Element von cTab1 (rekursiv) die Prozedur «A» auf.

PROCEDURE Delete(VAR E:cEle1);
Löscht das Element mit «E»-Key aus cTab1.

PROCEDURE Dispose;
Löscht alle Elemente aus cTab1.

PROCEDURE Empty():BOOLEAN;
Beantwortet im Funktionswert die Frage nach leerer cTab1.

PROCEDURE Find(VAR E:cEle1; Q:tQuality):BOOLEAN;
Sucht in cTab1 das Element welches den Bedingungen von «E».Key und «Q» entspricht und überträgt es im positiven Fall nach «E». Im Funktionswert wird die Aktion bestätigt.

PROCEDURE First(VAR E:cEle1):BOOLEAN;
Liefert in «E» das erste Element von cTab1. Die Aktion wird im Funktionswert bestätigt.

PROCEDURE Init;
Initialisert cTab1.

PROCEDURE Insert(VAR E:cEle1);
Fügt «E» in cTab1 ein. Achtung: bei doppelten Keys wird das bestehende Element überdeckt.

PROCEDURE Last(VAR E:cEle1):BOOLEAN;
Liefert in «E» das letzte Element von cTab1. Die Aktion wird im Funktionswert bestätigt.

PROCEDURE Replace(VAR E:cEle1);
Ersetzt ein bestehendes Element mit «E»-Key in cTab1 durch «E» (Key darf sich nicht verändert haben).

Tab_2

Einleitung

Dieser Modul realisiert eine Tabelle, deren interne Struktur ein ausgeglichener Baum (AVL balanced Tree) ist. Die Details können Niklaus Wirth's Buch «Algorithmen und Datenstrukturen mit Modula-2» entnommen werden.

Die Realisierung von solchen Bäumen basiert auf rekursiven Methoden welche zwar sehr effizient sind, aber den Nachteil haben, dass nach einem Unterbruch wieder beim Ausgangspunkt aufgesetzt werden muss. Um diesen Nachteil zu eliminieren, und trotzdem die Vorteile der rekursiven Methode nutzen zu können, sind die Elemente von Tab_2 zusätzlich verkettet und diesbezüglich gleichzeitig mit der Balancierung up-to-date.

Der Begriff «Key» steht jeweils als Synonym für die realen Schlüsseldaten in cEle2.

Beschreibung

```
TYPE
        tBalance = SHORTINT [-1..1];
        pcEle2 = POINTER TO cEle2;

        CLASS cEle2;
           Right : pcEle2;          (* Pointer to right sub-tree *)
           Left : pcEle2;           (* Pointer to left sub-tree  *)
           Bal  : tBalance;         (* Tree balance flag  *)
           Prev : pcEle2;           (* Pointer to previous Element *)
           Next : pcEle2;           (* Pointer to next Element *)
           VIRTUAL PROCEDURE Compare(p:pcEle2):INTEGER;
        END cEle2;
```

Bildet die Datenschnittstelle zu cTab2 für alle ihre Elemente die sie verwaltet. Die virtuelle Prozedur Compare muss überschrieben werden. Nur der Instanz welche cEle2 implementiert, sind ihre Daten bekannt.

```
        VIRTUAL PROCEDURE Compare(p:pcEle2):INTEGER;
```

Liefert als Funktionswert das Ergebnis aus dem Key-Vergleich von «cEle2» und «p». Für aufsteigend sortierte Elemente gilt: -1, 0, +1 (für kleiner, gleich, grösser).

```
TYPE
        tApply  = PROCEDURE(pcEle2);
```

Prozedur, welche die Daten von «pcEle2» verarbeitet.

```
        tQuality  = ARRAY[0..1]OF CHAR;
```

Definiert die Qualität des gesuchten, im Verhältnis zum angegebenen Key. Gültig sind: <, <=, =, >=, > (zweiwertige Symbole dürfen vertauscht werden).

```
        pcTab2 = POINTER TO cTab2;

        CLASS cTab2;
           Root : pcEle2;
           Top : pcEle2;
           Bottom : pcEle2;
           Curs : pcEle2;           (* cursor for Next,Prev*)
           Start : pcEle2;          (* for Read *)
           PROCEDURE Apply(A:tApply);
           PROCEDURE Delete(VAR E:cEle2);
           PROCEDURE Dispose;
           PROCEDURE Empty():BOOLEAN;
           PROCEDURE Find(VAR E:cEle2; Q:tQuality):pcEle2;
           PROCEDURE Find2(VAR E:cEle2; Q:tQuality):BOOLEAN;
           PROCEDURE First():pcEle2;
           PROCEDURE Init;
```

```
    PROCEDURE Insert(VAR E:cEle2);
    PROCEDURE Last():pcEle2;
    PROCEDURE Move(p:pcEle2; VAR E:cEle2):BOOLEAN;
    PROCEDURE Next():pcEle2;
    PROCEDURE Open(Forward:BOOLEAN):BOOLEAN;
    PROCEDURE Prev():pcEle2;
    PROCEDURE Read():pcEle2;
    PROCEDURE Replace(VAR E:cEle2);
END cTab2;
```

Die Ordnung der Elemente in cTab2 wird von cEle2.Compare determiniert. Diese Tatsache wird bei den einzelnen Prozeduren nicht mehr erwähnt, kann jedoch von Bedeutung sein, wenn z.B. durch Umkehrung des Compare.Funktionswertes absteigend sortiert wird.

PROCEDURE Apply(A:tApply);
Ruft mit jedem Element von cTab2 (rekursiv) die Prozedur «A» auf.

PROCEDURE Delete(VAR E:cEle2);
Löscht das Element mit «E»-Key aus cTab2.

PROCEDURE Dispose;
Löscht alle Elemente aus cTab2.

PROCEDURE Empty():BOOLEAN;
Beantwortet im Funktionswert die Frage nach leerer cTab2.

PROCEDURE Find(VAR E:cEle2; Q:tQuality):pcEle2;
Sucht in cTab2 das Element welches den Bedingungen von «E».Key und «Q» entspricht. Im Funktionswert wird der Pointer aus das Element, bzw. NIL zurückgeliefert.

PROCEDURE Find2(VAR E:cEle2; Q:tQuality):BOOLEAN;
Sucht in cTab2 das Element welches den Bedingungen von «E».Key und «Q» entspricht und überträgt es im positiven Fall nach «E». Im Funktionswert wird die Aktion bestätigt.

PROCEDURE First():pcEle2;
Liefert im Funktionswert den Pointer auf das erste Element, bzw. NIL.

PROCEDURE Init;
Initialisert cTab2.

PROCEDURE Insert(VAR E:cEle2);
Fügt «E» in cTab2 ein. Achtung: bei doppelten Keys wird das bestehende Element überdeckt.

PROCEDURE Last():pcEle2;
Liefert im Funktionswert den Pointer auf das letzte Element, bzw. NIL.

PROCEDURE Move(p:pcEle2; VAR E:cEle2):BOOLEAN;
Überträgt die Daten aus «p» nach «E». Sollte «p» NIL sein, wird «E» mit Null initalisiert. Im Funktionswert wird die Aktion bestätigt.

PROCEDURE Next():pcEle2;
Liefert im Funktionswert den Pointer auf das nächste Element.

PROCEDURE Open(Forward:BOOLEAN):BOOLEAN;
Setzt den Startpointer in cTab2 für Read() an den Anfang - wenn «Forward»=TRUE -, bzw. das Ende, - wenn «Forward»=FALSE- ist. Im Funktionswert wird die Aktion bestätigt.

PROCEDURE Prev():pcEle2;
Liefert im Funktionswert den Pointer auf das vorangehende Element.

PROCEDURE Read():pcEle2;
Jeder Aufruf liefert den Pointer auf das nächstfolgende Element, bzw. NIL, wenn Anfang oder Ende (Open) erreicht ist.

PROCEDURE Replace(VAR E:cEle2);

Ersetzt ein bestehendes Element mit «E»-Key in cTab2 durch «E» (Key darf sich nicht verändert haben).

Tex_1

Einleitung

Dieser Modul realisiert das dynamischeTextsystem. Es benötigt wenig Speicherplatz, da es im Prinzip Referenzen auf die Texte verwaltet, und erst, wenn ein Text (Prefix) erstmals abgerufen wird, diesen komprimiert speichert. Die Texte können mit einem beliebigen Editor für ASCII-Dateien gepflegt werden.

Die Definitionen in der Textdatei folgen dem Prinzip von Prefix und Suffix. An der Stellung des Dachsymbols erkennt man Prefix^ und ^Suffix. Zusammen bilden sie den Schlüssel (Key) für den Zugriff auf die Daten. Dabei findet keine Differenzierung nach Gross-/Kleinschrift und Umlautschreibweise statt. Zeilen ohne Prefix- und Suffix-Symbole sind lediglich Kommentar. Wird ein Schlüssel nicht gefunden, liefert das Textsystem den Schlüssel selbst, und zwar in der Form Prefix.Suffix. Das Tabulatorzeichen CHR(9) wird korrekt abgehandelt. Eine Zeile darf maximal 160 Stellen belegen, darüber wird abgeschnitten. Die Beispiele sind jeweils mit ♣ markiert.

1. Dateiname

Der Name muss so festgelegt werden, dass er (a) mit keiner bestehenden Datei *.TX* kollidiert und (b) als Dateiname vom DOS akzeptiert wird. Der Namensgeber-Prefix darf länger als der Dateiname sein. Aus «Beispielsweise^» entsteht somit die Datei «BEISPIEL.TX*», wobei * für die 1-te Stelle der Sprache steht (English, French,German,Italy,Spanish).

2. Prefix-Zeile

Der Prefix ist ein sprachunabhängiger Schlüssel. Alle nachfolgend definierten Suffixe sind ihm untergeordnet, bis ein weiterer Prefix folgt. In der Datei wird der Prefix vollständig angegeben: ♣ Beispielsweise^
In welcher Spalte er beginnt ist unwesentlich. Wichtig ist, dass keine weiteren Daten auf der selben Zeile stehen. In einer Datei dürfen mehrere Prefixe definiert werden. Der Dateiname muss jedoch vom Hauptprefix abgeleitet sein. Die andern sind versteckte Schlüssel und lassen sich nur indirekt finden. So muss z.B. in einer Modul-Hierarchie die höchste Ebene innerhalb der Datei als Namensgeber fungieren. Das Prinzip beim Suchen ist dabei folgendes: Ein dem System unbekannter Prefix wird als Datei gesucht. Ist die Datei gefunden, werden alle weiteren Prefixe in dieser Datei adressiert und sind damit bekannt.

```
♣ Beispielsweise^
        ^Suffix ...
        ...
  Versteckt^ ...
```

3. Suffix-Zeile

Der Suffix ist ein sprachunabhängiger Schlüssel. Er ist dem vorangehenden Prefix untergordnet. Die Zeile muss man sich in 3 Spalten aufgeteilt vorstellen: Suffix, Text und Schlüssel für Hilfe. Suffix und Text sind durch midestens eine Leerstelle getrennt. Jede weitere Leerstelle kann Einfluss auf die Anwendung haben. In welcher Spalte ein Suffix beginnt ist unwesentlich. DieVerwendung der Zeile ist aus der Definition selbst nicht ersichtlich. Dies ist im Programm festgelegt. Es ist deshalb möglich, dass die ganze Zeile als Text verwendet wird. Manchmal fehlt der Text. Er wurde nicht etwa vergessen, sondern bewusst ausgelassen. Dies für jene Fälle, wo im Programm ein Schlüssel existiert, und trotzdem kein Text angezeigt werden soll.

a) Text ohne Hilfe ♣ ^Suffix Text

Der Inhalt wird unverändert der Anwendung übergeben. Diese bestimmt die weitere Verarbeitung. Im Vordergrund stehen Überschriften für Fenster und Auswahlketten für die Map. Letztere enthalten eine Anzahl Elemente, die durch Kommas getrennt sind. Bei diesen sollten weder die Reihenfolge der Elemente, noch Kommas entfernt werden, auch dann nicht, wenn mehrere hintereinander stehen. Sie repräsentieren leere Elemente. Die Programme arbeiten mit der Ordnungszahl der Elemente, und nicht mit deren Namen. Mit geschweiften Klammern wird Auswahltext für Menus hervorgehoben ♣ Bei{s}piel.

b) Text mit Hilfe ♣ ^Suffix Text ^Hilfe

Mit geschweiften Klammern wird Text farblich hervorgehoben und, soweit es Keyboard-Funktionen betrifft, damit für die Maus ansprechbar. Mit «CHR(n)» für n=1..MAX(SHORTCARD) lassen sich Sonderzeichen dagestellen. Als Merkmal für die Verbindung zum Hilfesystem (Hlp_1) dient wieder das Dachsymbol ^. Es darf direkt am Text anschliessen, nicht aber an einen Suffix. Auch in den Hilfedateien werden Prefixe und Suffixe verwendet. DieVerbindung zu diesen erfolgt auf 4 verschiedene, nachfolgend definierte Arten (immer mit und ohne Text!):

wenn Prefixe und Suffixe identisch sind:

♣ ^SuffixTX text...^ ♣ ^SuffixTX ^

wenn nur die Prefixe identisch sind:
♣ ^SuffixTX text...^SuffixHP ♣ ^SuffixTX ^SuffixHP

wenn nur die Suffixe identisch sind:
♣ ^SuffixTX text...^PrefixHP. ♣ ^Suffix ^PrefixHP.

wenn weder Prefixe noch Suffixe identisch sind:
♣ ^SuffixTX text...^PrefixHP.SuffixHP ♣ ^SuffixTX text ...^PrefixHP.SuffixHP

Beschreibung

```
FROM FIOR IMPORT ExtStr;

TYPE
        tPos  = (Nothing,Left,Center,Right);
```

PROCEDURE Form(VAR Text:ARRAY OF CHAR; Body:CARDINAL; Pos:tPos);
Formt «Text» derart, dass er (a) in den Bereich von «Body» passt und (b) in diesem gemäss Pos» (Unverändert, Links, Mitte, Rechts) positioniert ist.

PROCEDURE SubstCHR(VAR Text:ARRAY OF CHAR);
Substituiert in «Text» alle 'CHR(n)' und 'chr(n)' mit n=1..MAX(SHORTCARD) durch das entsprechende Zeichen. So wird z.B aus CHR(27) das Zeichen → .

```
TYPE
        tText  = ARRAY[0..159] OF CHAR;
        prText;
        pcText = POINTER TO cText;

        CLASS cText;
           Text : prText;
           PROCEDURE Close;
           PROCEDURE Excl(Name:ARRAY OF CHAR);
           PROCEDURE Get(Key:ARRAY OF CHAR):tText;
           PROCEDURE Incl(Name:ARRAY OF CHAR):BOOLEAN;
           PROCEDURE Open(File:ARRAY OF CHAR):BOOLEAN;
        END cText;
```

Die wichtigsten Prozeduren im Ablauf sind: **Open**, **Get**.

PROCEDURE Close;
Gibt die Ressourcen der Klasse frei.

PROCEDURE Excl(Name:ARRAY OF CHAR);
Entfernt «Name» (Prefix) aus der Klasse.

PROCEDURE Get(Key:ARRAY OF CHAR):tText;
Liefert den Inhalt der Zeile mit «Key» (Prefix.Suffix). Ist keine Zeile vorhanden, wird der «Key» geliefert. Ansonsten versteht sich Inhalt ohne Key.

PROCEDURE Incl(Name:ARRAY OF CHAR):BOOLEAN;
Installiert «Name» (Prefix) in der Klasse.

PROCEDURE Open(File:ARRAY OF CHAR):BOOLEAN;
Öffnet die Klasse mit «File» als Einstiegsdatei. Normalerweise mit gleichem Stammnamen wie die EXE-Datei (Modul.TX* = Modul.EXE).

Tim_1

Einleitung

Dieser Modul stellt Funktionen für alle Operationen mit der Zeit zur Verfügung.

Die Aufbereitung von extern nach intern, bzw. umgekehrt, erfolgt jeweils anhand einer Maske. Diese definiert die Zeit in beliebiger Zusammensetzung und Kombination von HH=Stunde; MM=Minute; SS=Sekunde und CC=Hundertstelsekunde. Als Interpunktionszeichen dienen Doppelpunkt, Punkt, Binde- und Schrägstrich. Anstelle von «HMSC» ist auch «hmsc» zulässig. Z.B. 'HH:MM:SS:CC' oder 'hh:mm:ss:cc' .

Beschreibung

```
TYPE
        rTime = RECORD
            Hrs   : SHORTCARD;
            Mins  : SHORTCARD;
            Secs  : SHORTCARD;
            Hsecs : SHORTCARD;
        END;

        tTimeFormat = (HMSC);
        rTimeFormat = RECORD
            Format : tTimeFormat;
            Mark   : CHAR;
        END;

VAR
        TimeFormat : rTimeFormat;
```

PROCEDURE CharTime(Char,Mask:ARRAY OF CHAR):rTime;
Extrahiert aus «Char» die Zeit und stellt sie im Funktionswert zur Verfügung. Über «Mask» wird die Behandlung gesteuert (Einleitung).

PROCEDURE Compare(Time1,Time2:rTime):INTEGER;
Vergleicht «Time1» mit «Time2» und gibt als Ergebnis -1, 0, +1 (für kleiner , gleich, grösser) zurück.

PROCEDURE Differ(Time1,Time2:rTime):rTime;
Errechnet die Differenz zwischen «Time1» und «Time2» und gibt das Ergebnis im Funktionswert zurück. In welchem Parameter welche Zeit steht, spielt keine Rolle. Die Funktion testet selber, welche grösser, und welche kleiner ist.

PROCEDURE GetSystem():rTime;
Liefert im Funktionswert die Systemzeit.

PROCEDURE SplitMask(VAR Mask,Hrs,Mins,Secs,Hsecs:ARRAY OF CHAR):BOOLEAN;
Splittet «Mask» (Einleitung) in die Bestandteile «Hrs», «Mins», «Secs» und «Hsecs» auf.

PROCEDURE TimeChar(Time:rTime; Mask:ARRAY OF CHAR;VAR Char:ARRAY OF CHAR);
Bereitet aus «Time» und «Mask» (Einleitung) die externe Zeit auf und stellt sie in «Char».

PROCEDURE Verify(Time:rTime):BOOLEAN;
Prüft «Time» und gibt das Ergebnis als Funktionswert zuück.

Beispielprogramme «Swiss Lotto»

CHL1DB: Database

```
DEFINITION MODULE CHL1DB;
IMPORT Btree;
FROM Btree IMPORT FHandle, IHandle;
FROM Dat_1 IMPORT rDate, tDayOfWeek;
FROM Tex_1 IMPORT tText, pcText;
FROM Scr_1 IMPORT rPal;
FROM FIO_1 IMPORT tPath;
TYPE
  sDrawDays = SET OF tDayOfWeek;
CONST
  MinTips = 2; MaxTips = 14;
  Quantity = 45; Extract = 6;
  DrawDays = sDrawDays{Wednesday,Saturday};
TYPE
  rHandle = RECORD
    File     : FHandle;
    Tip      : IHandle;
    TipX     : IHandle;
    Draw     : IHandle;
    DrawX    : IHandle;
    Result   : IHandle;
    ResultX  : IHandle;
    pTex     : pcText;
  END;
  tTip = ARRAY[1..Extract]OF SHORTCARD;
  rTipKey = RECORD
    Date : rDate;
    Time : LONGCARD;
  END;
  rTip = RECORD
    Key   : rTipKey;
    Weeks : SHORTCARD;
    Joker : LONGCARD;
    Tip   : ARRAY[1..MaxTips]OF tTip;
  END;
  rDraw = RECORD
    Key   : rDate;
    Joker : LONGCARD;
    Draw  : ARRAY[1..Extract+1]OF SHORTCARD;
  END;
  rResult = RECORD
    Draw : rDate;
    Tip  : rTipKey;
  END;

CLASS cDB;
  H : rHandle;
  PROCEDURE AppendDraw(VAR Rec:rDraw):BOOLEAN;
  PROCEDURE AppendTip(VAR Rec:rTip):BOOLEAN;
  PROCEDURE ChangeDraw(VAR Rec:rDraw):BOOLEAN;
  PROCEDURE ChangeTip(VAR Rec:rTip):BOOLEAN;
  PROCEDURE Close;
  PROCEDURE FindIndex(I:IHandle; Key:ARRAY OF BYTE;VAR Loc:LONGCARD):BOOLEAN;
  PROCEDURE GetDraw(Key:rDate; VAR Rec:rDraw):BOOLEAN;
  PROCEDURE GetTip(Key:rTipKey; VAR Rec:rTip):BOOLEAN;
  PROCEDURE Next(I:IHandle; VAR Rec:ARRAY OF BYTE):BOOLEAN;
  PROCEDURE NextIndex(I:IHandle; VAR Loc:LONGCARD):BOOLEAN;
  PROCEDURE Open(FP:ARRAY OF CHAR; RdOnly,Share,Create:BOOLEAN; Tex:pcText;
                 Pal:rPal);
  PROCEDURE Prev(I:IHandle; VAR Rec:ARRAY OF BYTE):BOOLEAN;
  PROCEDURE PrevIndex(I:IHandle; VAR Loc:LONGCARD):BOOLEAN;
  PROCEDURE RemoveDraw(Key:rDate);
  PROCEDURE RemoveTip(Key:rTipKey);
  PROCEDURE Reset(I:IHandle);
  PROCEDURE Test(FP:ARRAY OF CHAR; RdOnly,Share:BOOLEAN; Tex:pcText;
                 Pal:rPal):BOOLEAN;
```

```
END cDB;

PROCEDURE SortTip(VAR Tip:tTip);

END CHL1DB.
```

CHL1: Main

```
MODULE CHL1;
IMPORT DOS_1, Scr_2, FIO, CHL11, CHL12, CHL13, CHL14;
FROM Head_1 IMPORT cHead;
FROM Scr_1 IMPORT cL1;
FROM CHL10 IMPORT DAT, Pal, Tex, DB;
VAR
  Head  : cHead;
  L1    : cL1;                                  ♣ Scr_1.cL1
  kb    : CHAR;
  pick  : SHORTCARD;

  PROCEDURE _ExistsDB():BOOLEAN;
  BEGIN
    CASE DB.H.File = NIL OF
    | TRUE  : CASE DB.Test(DAT.File,FALSE,TRUE,ADR(Tex),Pal('CHL1.L1')^) OF
              | TRUE  : DB.Open(DAT.File,FALSE,TRUE,FALSE,
                                ADR(Tex),Pal('CHL1.L1')^);
                        RETURN(TRUE);
              | FALSE : L1.SetPickNr(3);
                        RETURN(FALSE);
              END;
    | FALSE : RETURN(TRUE);
    END;
  END _ExistsDB;
BEGIN
  Head.Open(Tex.Get('CHL1.Head'),'4.0','1996',ADR(Tex));
  WITH L1 DO
    Open(Tex.Get('CHL1.L25'),Pal('CHL1.L1')^,ADR(Tex));
    Add(Tex.Get('CHL1.Tip'),1);
    Add(Tex.Get('CHL1.Draw'),2);
    Add(Tex.Get('CHL1.Result'),6);
    Add(Tex.Get('CHL1.Base'),3);
    Add(Tex.Get('CHL1.Setup'),4);
    Add(Tex.Get('CHL1.DOS'),5);
    LOOP
      Head.Put;
      pick:=Pick(kb);
      Head.Clear;
      CASE pick OF
      | 1  : IF _ExistsDB() THEN
               CHL12.Tip;
             END;
      | 2  : IF _ExistsDB() THEN
               CHL13.Draw;
             END;
      | 3  : CHL11.Base;
      | 4  : Scr_2.Setup(ADR(Tex),DAT.Pals,L1);
      | 5  : IF NOT DOS_1.Command() THEN
               PutError(Tex.Get('CHL1.L24NoDOS'));
             END;
      | 6  : IF _ExistsDB() THEN
               CHL14.Result;
             END;
      | ELSE HALT;
      END;
    END;
  END;
END CHL1.
```

CHL10: Common for main

```
DEFINITION MODULE CHL10;
IMPORT FIO;
FROM FIO_1 IMPORT tPath;
FROM Tex_1 IMPORT cText;
FROM Scr_1 IMPORT rPal, prPal;
FROM CHL1DB IMPORT cDB;
TYPE
  rDAT = RECORD
    Pals  : ARRAY[0..29] OF rPal;
    File  : tPath;
    Draws : CARDINAL;
  END;
VAR
  DAT  : rDAT;
  Tex  : cText;
  DB   : cDB;

PROCEDURE Exit;

PROCEDURE Pal(For:ARRAY OF CHAR):prPal;

END CHL10.
```

```
IMPLEMENTATION MODULE CHL10;
IMPORT FIO_1, Str, Lib, Scr_1, Scr_4;
FROM Scr_1 IMPORT AltXProc, AltXHalt, User;
FROM FIO_1 IMPORT tName;
VAR
  exit : PROC;

PROCEDURE _RdDAT;
VAR
  file : tName;
  ok : BOOLEAN;
BEGIN
  ok:=FIO_1.RdDAT(User.Ref,'CHL1',DAT);
  Str.Copy(file,'CHL1.TX'); Str.Append(file,User.Lang);
  ok:=Tex.Open(file);
  Str.Copy(file,'CHL1.HP'); Str.Append(file,User.Lang);
  Scr_1.InitGlobal(ADR(Tex),file);
END _RdDAT;

PROCEDURE Exit;
BEGIN
  IF DB.H.File # NIL THEN
    DB.Close;
  END;
  FIO_1.WrDAT(DAT);
  Scr_4.CloseAll;
  exit;
END Exit;

PROCEDURE Pal(For:ARRAY OF CHAR):prPal;
BEGIN
  RETURN(Scr_1.GetPal(For,DAT.Pals));
END Pal;

BEGIN
  AltXProc:=AltXHalt;
  _RdDAT;
  Lib.Terminate(Exit,exit);
END CHL10.
```

Datei CHL1.TXG

```
CHL1^
  ^Base {B}asis
  ^DOS {D}OS
  ^Draw {Z}iehung
  ^Head Swiss Lotto
  ^L24NoDOS  DOS konnte nicht geladen werden^DOS_1.L24
  ^L25  {↵}=Wahl {Esc}=Abbruch {AltX}=Exit {F1}=Hilfe^
  ^Result {R}esultat
  ^Setup {O}ptionen
  ^Tip {T}ip
```

Datei CHL1.HPG

```
CHL1^ SwissLotto
  ^Basis=CHL11.M1L25
  ^Bildschirm=Scr_1.?
  ^DOS DOS
     Dieser Punkt verzweigt zum Betriebssystem. Dort musst
     Du, um zurückzukehren, EXIT eingeben.
  ^Hauptmenu=Scr_1.L1
  ^L25 CHL1.EXE
     Generiert und verwaltet Lottoscheine. Nach Eingabe
     der Ziehung kannst Du Deine Resultate abfragen.
     Fenstertyp:{Hauptmenu}{Bildschirm}
     Themen:{Tip}{Ziehung}{Resultat}{Basis}{Optionen}{DOS}
  ^Optionen=Scr_2.?
  ^Resultat=CHL14.M1L25
  ^Tip=CHL12.L25
  ^Ziehung=CHL13.L25
```

CHL13: Draw

```
  DEFINITION MODULE CHL13;
  PROCEDURE Draw;
  END CHL13.
```

```
  IMPLEMENTATION MODULE CHL13;
  IMPORT Lib, Btree, CHL131, CHL132, CHL133;
  FROM CHL10 IMPORT DAT, Pal, Tex, DB;
  FROM CHL1DB IMPORT rDate;
  FROM Scr_1 IMPORT cPop;
  VAR
    key : rDate;

  PROCEDURE Draw;
  VAR
    Pop  : cPop;                                  ♣ Scr_1.cPop
    Kb   : CHAR;

    PROCEDURE EmptyDB():BOOLEAN;
    BEGIN
      WHILE NOT Btree.LockIHandle(DB.H.Draw) DO
      END;
      CASE Btree.RecordCount(DB.H.Draw) = 0 OF
      | TRUE  : Btree.UnLockIHandle(DB.H.Draw);
                Pop.PutError(Tex.Get('CHL13.NoData'));
                RETURN(TRUE);
      | FALSE : Btree.UnLockIHandle(DB.H.Draw);
                RETURN(FALSE);
      END;
    END EmptyDB;
  BEGIN
    WITH Pop DO
      Open(Tex.Get('CHL13.Draw'),Tex.Get('CHL13.L25'),Pal('CHL13.Draw')^,ADR(Tex));
      Add(Tex.Get('CHL13.Append'),1);
      Add(Tex.Get('CHL13.Browse'),2);
```

```
      Add(Tex.Get('CHL13.Statistics'),3);
      LOOP
        CASE Pick(Kb) OF
        | 1 : CHL131.Modify(key);
        | 2 : IF NOT EmptyDB() THEN
                CHL132.Browse;
              END;
        | 3 : IF NOT EmptyDB() THEN
                CHL133.Statistics;
              END;
        | ELSE EXIT;
        END;
      END;
      Close;
    END;
  END Draw;

  BEGIN
    Lib.Fill(ADR(key),SIZE(key),0);
  END CHL13.
```

Datei CHL13.TXG

```
CHL13^
  ^Append {H}inzufügen
  ^Browse {D}urchblättern
  ^Statistics {S}tatistik
  ^L25  {↵}=Wahl {Esc}=Abbruch {AltX}=Exit {F1}=Hilfe^
  ^NoData  Datenbank ohne Daten^
  ^Draw
```

Datei CHL13.HPG

```
CHL13^ Ziehung
  ^Bildschirm=Scr_1.?
  ^Durchblättern=CHL132.M1L25
  ^Hinzufügen=CHL131.M1L25
  ^L25 Ziehung
     Hier werden die Ziehungen verwaltet.
     Fenstertyp:{Untermenu}{Bildschirm}
     Themen:{Hinzufügen}{Durchblättern}{Statistik}
  ^NoData
     Datenbank enthält noch keine Ziehung.
  ^Untermenu=Scr_1.PopUp
  ^Statistik=CHL133.M1L25
```

CHL131: Modify Draw

```
  DEFINITION MODULE CHL131;
  FROM CHL1DB IMPORT rDate;
  PROCEDURE Modify(Key:rDate);
  END CHL131.


  IMPLEMENTATION MODULE CHL131;
  IMPORT Dat_1, Lib, Com_1, Str, CHL1DB;
  FROM CHL10 IMPORT DAT, Pal, Tex, DB;
  FROM CHL1DB IMPORT rDraw, DrawDays, Quantity, tTip;
  FROM Com_1 IMPORT PicRPZ;
  FROM Tex_1 IMPORT tText, tPos;
  FROM Scr_1 IMPORT cMap, sKb, sAttr, Enter, Enter3, Ground0, Skip, Skip4, F7;
  VAR
    M1   : cMap;                                  ♣ Scr_1.cMap
    Kb   : CHAR;
    Kbs  : sKb;
    Upd  : BOOLEAN;
    add  : BOOLEAN;
    draw : rDraw;
    n,x  : CARDINAL;

  PROCEDURE _UpdDate(Adr:ADDRESS; VAR Kb:CHAR; VAR Put:BOOLEAN);
  VAR
```

```
  d : rDraw;
BEGIN
  WITH draw DO
    WHILE NOT (Dat_1.WhichDayOfWeek(Key) IN DrawDays) DO
      Key:=Dat_1.Modify(Key,+1); Put:=TRUE;
    END;
    IF DB.GetDraw(Key,d) THEN
      M1.PutErrorAdr(Adr,Tex.Get('CHL131.M1FindDraw'));
    END;
  END;
END _UpdDate;

PROCEDURE _UpdDraw(Adr:ADDRESS; VAR Kb:CHAR; VAR Put:BOOLEAN);
VAR
  sc : POINTER TO SHORTCARD;
BEGIN
  sc:=Adr;
  CASE (sc^ = 0)
    OR (sc^ > Quantity) OF
  | TRUE  : M1.PutErrorAdr(Adr,Tex.Get('CHL131.M1NoDraw'));
  | FALSE : WITH draw DO
              FOR n:=1 TO HIGH(Draw) DO
                IF (ADR(Draw[n]) # Adr)
                  & (sc^ = Draw[n]) THEN
                  M1.PutErrorAdr(Adr,Tex.Get('CHL131.M1DupDraw'));
                END;
              END;
            END;
  END;
END _UpdDraw;

PROCEDURE _UpdJoker(Adr:ADDRESS; VAR Kb:CHAR; VAR Put:BOOLEAN);
BEGIN
  IF draw.Joker = 0 THEN
    M1.PutErrorAdr(Adr,Tex.Get('CHL131.M1NoJoker'));
  END;
END _UpdJoker;


PROCEDURE Modify(Key:rDate);
BEGIN
  add:=NOT DB.GetDraw(Key,draw);
  WITH M1 DO  WITH draw DO
    IF add THEN
      Lib.Fill(ADR(draw),SIZE(draw),0);
    END;
    Open(0,1,43,4,TRUE,Tex.Get('CHL131.M1'),
         Tex.Get('CHL131.M1L25'),Pal('CHL131.M1')^,ADR(Tex));
    AddText(Tex.Get('CHL131.M1L1'),2,1,40,Nothing,Ground0);
    AddText('-',30,2,1,Nothing,Ground0);
    CASE add OF
    | TRUE  : AddDate(Key,2,2,'DD.MM.YY',sAttr{},Enter3,_UpdDate,
                      NULLPROC,sKb{},Tex.Get('CHL131.M1L25Date'));
              SetErrorAdr(ADR(Key),TRUE);
    | FALSE : AddDate(Key,2,2,'DD.MM.YY',sAttr{Skip},Skip4,NULLPROC,
                      NULLPROC,sKb{},Tex.Get('CHL131.M1L25Date'));
    END;
    x:=12;
    FOR n:=1 TO HIGH(Draw) DO
      AddShortCard(Draw[n],x,2,2,PicRPZ,sAttr{},Enter3,_UpdDraw,
                   NULLPROC,sKb{},Tex.Get('CHL131.M1L25Draw'));
      IF add THEN
        SetErrorAdr(ADR(Draw[n]),TRUE);
      END;
      CASE n =HIGH(Draw)-1 OF
      | TRUE  : x:=x+5;
      | FALSE : x:=x+3;
      END;
    END;
    AddLongCard(Joker,36,2,6,PicRPZ,sAttr{},Enter3,_UpdJoker,
                NULLPROC,sKb{},Tex.Get('CHL131.M1L25Joker'));
```

```
      IF add THEN
        SetErrorAdr(ADR(Joker),TRUE);
      END;
      GotoHome;
      SetOkMsg(F7,Tex.Get('CHL131.M1F7'));
      Kbs:=sKb{Enter};
      LOOP
        Get(Kbs,Kb,Upd);
        CASE Kb OF
        | Enter : INCL(Kbs,F7);
        | F7    : CHL1DB.SortTip(tTip(ADR(Draw)^));
                  CASE add OF
                  | TRUE  : IF DB.AppendDraw(draw) THEN
                              EXIT;
                            END;
                  | FALSE : IF DB.ChangeDraw(draw) THEN
                              EXIT;
                            END;
                  END;
                  Refresh;
        | ELSE EXIT;
        END;
      END;
      Close;
    END;  END;
  END Modify;

  BEGIN
  END CHL131.
```

Datei CHL131.TXG

```
CHL131^
  ^M1 Ziehung
  ^M1L1 Datum     Zahlen                    Joker
  ^M1L25  {F1}=Hilfe {Esc}=Zurück {AltX}=Exit {↵}=ok^
  ^M1L25Date ^
  ^M1L25Draw ^
  ^M1L25Joker ^
  ^M1F7  {F7}=Speichern
  ^M1FindDraw  Ziehung für dieses Datum bereits vorhanden^
  ^M1NoDraw  Zahl muss zwischen 1 und 45 liegen^
  ^M1DupDraw  Doppelte Zahl^
  ^M1NoJoker  Jokerzahl fehlt^
```

Datei CHL131.HPG

```
CHL131^ Ziehung hinzufügen
  ^Bildschirm=Scr_1.?
  ^Durchblättern=CHL132.M1L25
  ^Map=Scr_1.Map
  ^M1DupDraw
     Eine Zahl darf pro Ziehung nur einmal vorkommen.
  ^M1FindDraw
     Für dieses Datum ist bereits eine Ziehung erfasst.
  ^M1L25 Hinzufügen
     Hier erfasst Du die Ziehungen, änderst oder entfernst
     sie beim {Durchblättern}.
     Fenstertyp:{Map}{Bildschirm}
     Themen:{Datum}{Zahlen}{Joker}
  ^M1L25Date Datum
     Hier wird das Datum der Ziehung erfasst.
     Thema:{Hinzufügen}
  ^M1L25Joker Joker
     Hier wird die Jokerzahl erfasst.
     Thema:{Hinzufügen}
  ^M1L25Draw Zahlen
     Hier erfasst Du die gezogenen Zahlen und zuletzt die
     Zusatzzahl.
     Thema:{Hinzufügen}
  ^M1NoDraw
```

```
   Zahl muss zwischen 1 und 45 liegen
^M1NoJoker
   Jokerzahl muss immer erfasst werden.
```

CHL132: Browse Draw

```
DEFINITION MODULE CHL132;
PROCEDURE Browse;
END CHL132.


IMPLEMENTATION MODULE CHL132;
IMPORT Lib, Dat_1, CHL131, Scr_1;
FROM CHL10 IMPORT DAT, Pal, Tex, DB;
FROM CHL1DB IMPORT rDraw, Extract;
FROM Com_1 IMPORT PicRPZ;
FROM Tex_1 IMPORT tText, tPos;
FROM Scr_1 IMPORT cMap, sKb, sAttr, Ground0, Skip, Skip4, Enter, F2,
  F3, ArrUp, ArrDn, PgUp, PgDn;
PROCEDURE _UpDn(Area:ADDRESS; CursY:CARDINAL; Kb:CHAR); FORWARD;
VAR
  M1    : cMap;                                  ♣ Scr_1.cMap
  M1Arr : ARRAY[1..12]OF rDraw;
  Kb    : CHAR;
  Kbs   : sKb;
  Upd   : BOOLEAN;
  n,x,y : CARDINAL;
  TopDraw : rDraw;

PROCEDURE _LoadArea(Area:ADDRESS; Q:CHAR);
VAR
  y : CARDINAL;
  loc : LONGCARD;
  rec : rDraw;
BEGIN
  y:=1;
  WHILE (y <= HIGH(M1Arr)) & (M1Arr[y].Key.Year # 0) DO
    INC(y);
  END;
  CASE (Q = '>') OF
  | TRUE  : WHILE (y <= HIGH(M1Arr))
              & (DB.Next(DB.H.DrawX,M1Arr[y])) DO
              INC(y);
            END;
            CASE (M1Arr[1].Key.Year = 0)
              OR (M1Arr[1].Key = TopDraw.Key) OF
            | TRUE  : Kbs:=Kbs - sKb{PgUp,ArrUp};
            | FALSE : Kbs:=Kbs + sKb{PgUp,ArrUp};
            END;
            CASE DB.NextIndex(DB.H.DrawX,loc) OF
            | TRUE  : Kbs:=Kbs + sKb{PgDn,ArrDn};
            | FALSE : Kbs:=Kbs - sKb{PgDn,ArrDn};
            END;
  | FALSE : Kbs:=Kbs + sKb{PgDn,ArrDn};
            WHILE (y <= HIGH(M1Arr))
              & (DB.Prev(DB.H.DrawX,rec)) DO
              Lib.Move(ADR(M1Arr[1]),ADR(M1Arr[2]),
                       SIZE(M1Arr)-SIZE(M1Arr[1]));
              M1Arr[1]:=rec;
              INC(y);
            END;
            CASE DB.PrevIndex(DB.H.DrawX,loc) OF
            | TRUE  : Kbs:=Kbs + sKb{PgUp,ArrUp};
            | FALSE : Kbs:=Kbs - sKb{PgUp,ArrUp};
            END;
  END;
  WITH M1 DO
    DEC(y); SetArea(Area,y,Kbs,'');          ♣ Scr_1.cMap.SetArea
    FOR y:=y TO 1 BY -1 DO
      WITH M1Arr[y] DO
```

```
        AddText('/',30,y+1,1,Nothing,Ground0);
      END;
    END;
  END;
END _LoadArea;

PROCEDURE _Other(Area:ADDRESS; y:CARDINAL; Kb:CHAR);
BEGIN
  WITH M1Arr[y] DO
    CASE Kb OF
    | F2    : CHL131.Modify(Key);
              IF DB.GetDraw(Key,M1Arr[y]) THEN
                M1.Refresh;
                M1.GotoAdr(ADR(M1Arr[y]));
              END;
    | F3    : IF Scr_1.ReplyKey(Tex.Get('CHL132.M1Delete'),
                       sKb{},Pal('CHL132.M1')^) = Enter THEN
                DB.RemoveDraw(Key);
                _UpDn(Area,y,'');
              END;
    END;
  END;
END _Other;

PROCEDURE _UpDn(Area:ADDRESS; CursY:CARDINAL; Kb:CHAR);
VAR
  y : CARDINAL;
  loc : LONGCARD;
  ok : BOOLEAN;
BEGIN
  FOR y:=1 TO HIGH(M1Arr) DO
    WITH M1Arr[y] DO  WITH M1 DO
      AddText('',30,y+1,1,Nothing,Ground0);
    END;  END;
  END;
  CASE Kb OF
  | PgDn :
      ok:=DB.FindIndex(DB.H.DrawX,M1Arr[HIGH(M1Arr)],loc);
      CASE CursY < 2 OF
      | TRUE  : Lib.Fill(ADR(M1Arr),SIZE(M1Arr),0);
      | FALSE : FOR y:=CursY TO HIGH(M1Arr) DO
                  M1Arr[y-CursY+1]:=M1Arr[y];
                  Lib.Fill(ADR(M1Arr[y]),SIZE(M1Arr[y]),0);
                END;
      END;
      _LoadArea(Area,'>');
      M1.GotoAdr(ADR(M1Arr[1]));
  | ArrDn :
      ok:=DB.FindIndex(DB.H.DrawX,M1Arr[HIGH(M1Arr)],loc);
      FOR y:=2 TO HIGH(M1Arr) DO
        M1Arr[y-1]:=M1Arr[y];
      END;
      Lib.Fill(ADR(M1Arr[HIGH(M1Arr)]),SIZE(M1Arr[HIGH(M1Arr)]),0);
      _LoadArea(Area,'>');
      M1.GotoAdr(ADR(M1Arr[HIGH(M1Arr)]));
  | PgUp     :
      ok:=DB.FindIndex(DB.H.DrawX,M1Arr[1],loc);
      CASE CursY = HIGH(M1Arr) OF
      | TRUE  : Lib.Fill(ADR(M1Arr),SIZE(M1Arr),0);
      | FALSE : FOR y:=CursY+1 TO HIGH(M1Arr) DO
                  Lib.Fill(ADR(M1Arr[y]),SIZE(M1Arr[y]),0);
                END;
      END;
      _LoadArea(Area,'<');
      y:=HIGH(M1Arr);
      WHILE (y > 1) & (M1Arr[y].Key.Year = 0) DO
        DEC(y);
      END;
      IF y # HIGH(M1Arr) THEN
        ok:=DB.FindIndex(DB.H.DrawX,M1Arr[y],loc);
        _LoadArea(Area,'>');
```

```
      END;
      M1.GotoAdr(ADR(M1Arr[y]));
  | ArrUp :
      ok:=DB.FindIndex(DB.H.DrawX,M1Arr[1],loc);
      Lib.Fill(ADR(M1Arr[HIGH(M1Arr)]),SIZE(M1Arr[HIGH(M1Arr)]),0);
      _LoadArea(Area,'<');
      M1.GotoAdr(ADR(M1Arr[1]));
  | ELSE (* delete *)
      IF CursY < HIGH(M1Arr) THEN
        FOR y:=CursY+1 TO HIGH(M1Arr) DO
          M1Arr[y-1]:=M1Arr[y];
        END;
      END;
      y:=HIGH(M1Arr); Lib.Fill(ADR(M1Arr[y]),SIZE(M1Arr[y]),0);
      WHILE (y > 1) & (M1Arr[y].Key.Year = 0) DO
        DEC(y);
      END;
      ok:=DB.FindIndex(DB.H.DrawX,M1Arr[y],loc);
      _LoadArea(Area,'>');
      M1.GotoAdr(ADR(M1Arr[CursY]));
  END;
END _UpDn;

PROCEDURE Browse;
BEGIN
  DB.Reset(DB.H.DrawX);
  IF NOT DB.Next(DB.H.DrawX,TopDraw) THEN
    Kb:=Scr_1.ReplyKey(Tex.Get('CHL132.M1NoDraw'),sKb{},
                  Pal('CHL132.M1')^);
    RETURN;
  END;
  DB.Reset(DB.H.DrawX);
  Lib.Fill(ADR(M1Arr),SIZE(M1Arr),0);
  WITH M1 DO
    Open(0,1,43,15,TRUE,Tex.Get('CHL132.M1'),
         Tex.Get('CHL132.M1L25'),Pal('CHL132.M1')^,ADR(Tex));
    AddText(Tex.Get('CHL132.M1L1'),2,1,40,Nothing,Ground0);
    FOR y:=1 TO HIGH(M1Arr) DO
      WITH M1Arr[y] DO
        AddDate(Key,2,y+1,'DD.MM.YY',sAttr{Skip},Skip4,
                NULLPROC,NULLPROC,sKb{},'');
        AddText('/',30,y+1,1,Nothing,Ground0);
        x:=12;
        FOR n:=1 TO HIGH(Draw) DO
          AddShortCard(Draw[n],x,y+1,2,PicRPZ,sAttr{Skip},Skip4,
                       NULLPROC,NULLPROC,sKb{},'');
          CASE n =HIGH(Draw)-1 OF
          | TRUE  : x:=x+5;
          | FALSE : x:=x+3;
          END;
        END;
        AddLongCard(Joker,36,y+1,6,PicRPZ,sAttr{Skip},Skip4,
                    NULLPROC,NULLPROC,sKb{},'');
      END;
    END;
    Kbs:=sKb{F2,F3};
    _LoadArea(AddArea(1,2,42,HIGH(M1Arr)+1,_UpDn,_Other),'>'); ♣ Scr_1.cMap.AddArea
    GotoAdr(ADR(M1Arr[1].Key));
    Get(sKb{},Kb,Upd);
    Close;
  END;
END Browse;

BEGIN
END CHL132.
```

Datei CHL132.TXG

```
CHL132^
  ^M1 Ziehung
  ^M1Delete  {↵}=Löschen
  ^M1L1 Datum     Zahlen                    Joker
  ^M1L25  {F1}=Hilfe {Esc}=Zurück {AltX}=Exit {↵}=ok {F2}=Ändern {F3}=Löschen^
  ^M1NoDraw  Keine Ziehungen in DB. Weiter mit{↵}^
```

Datei CHL132.HPG

```
CHL132^ Durchblättern
  ^Bildschirm=Scr_1.?
  ^Map=Scr_1.Map
  ^M1L25 Durchblättern
     Hier kannst Du die Ziehungen durchblättern, mit [F2]
     ändern und mit [F3] löschen (muss bestätigt werden).
     Fenstertyp:{Map}{Bildschirm}
  ^M1NoDraw
     Die Datenbank enthält keine Ziehung mehr.
```

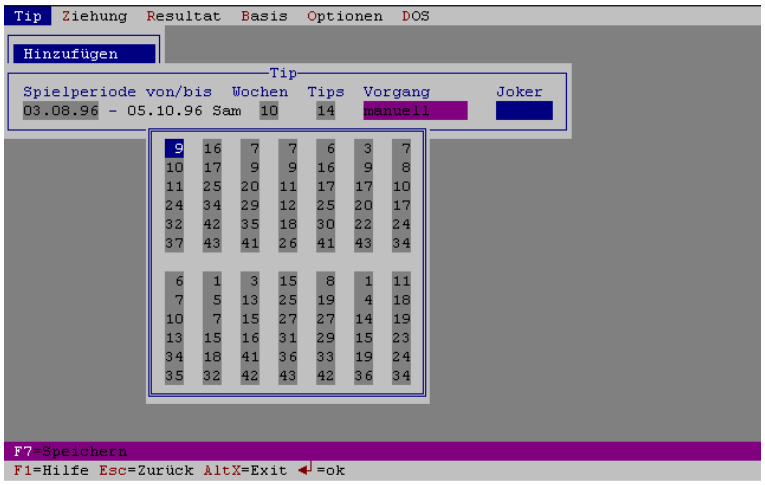
Endprodukt «Swiss Lotto»

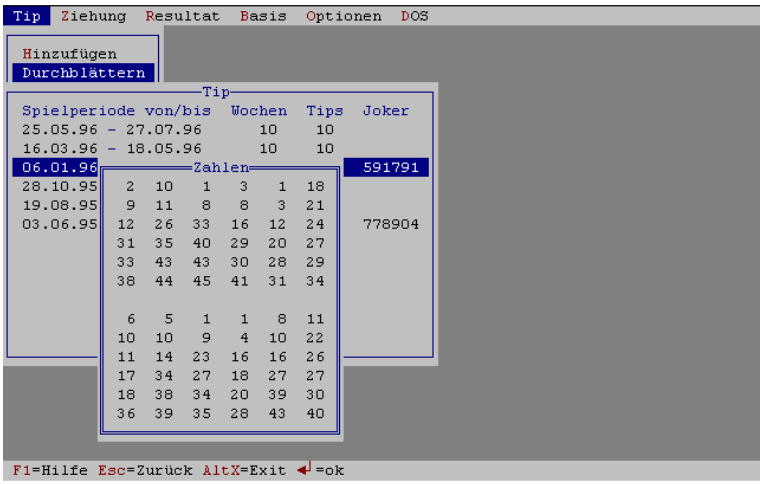
Mit «Swiss Lotto» lassen sich Tips sowohl erfassen, als auch automatisch generieren. Für erfasste Ziehungen sind die Resultate auswertbar.

Über den Menupunkt «Optionen» können verschiedene Kriterien den persönlichen Bedürfnissen angepasst werden. Um die Farbgebung eines Bildes ändern zu können, muss darin [AltS] betätigt werden. Auf den Fehler- und Hinweiszeilen (Zeilen 24/25) verhält sich die Maus textsensitiv. Dort aufgeführte Funktionen reagieren damit auf Mausklicks.

Tip

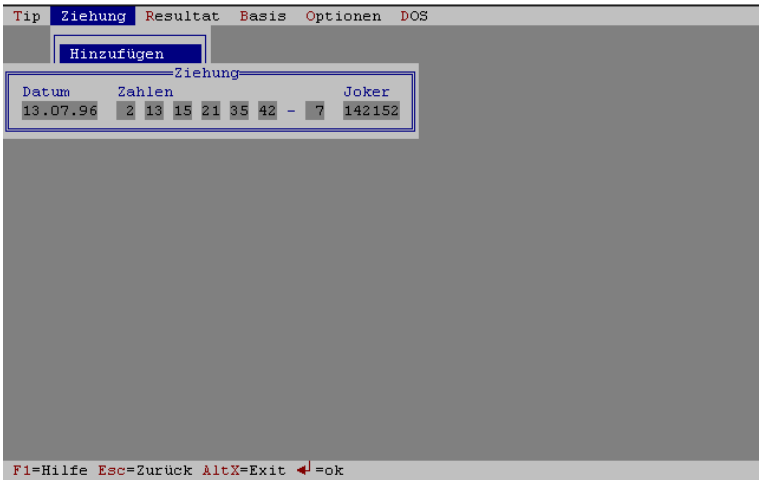
Tips lassen sich auf drei verschiedene Arten erstellen: (a) manuell, (b) automatisch aus Basismenge 1..45 und (c) automatisch aus Basismenge und allen bisher gezogenen Zahlen. Das Ergebnis darf beliebig verworfen und abgeändert werden. Die Jokerzahl sollte spätestens bei der Resultatauswertung eingetragen sein.





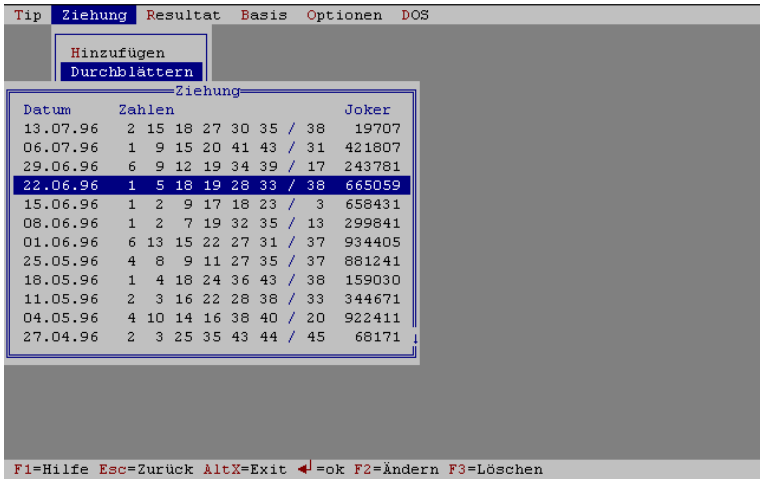
Tips lassen sich durchblättern. Mit [↵] wird die Zahlenreihe angezeigt, [F2] wechselt zum Mutationsbild und [F3] löscht die Zeile.

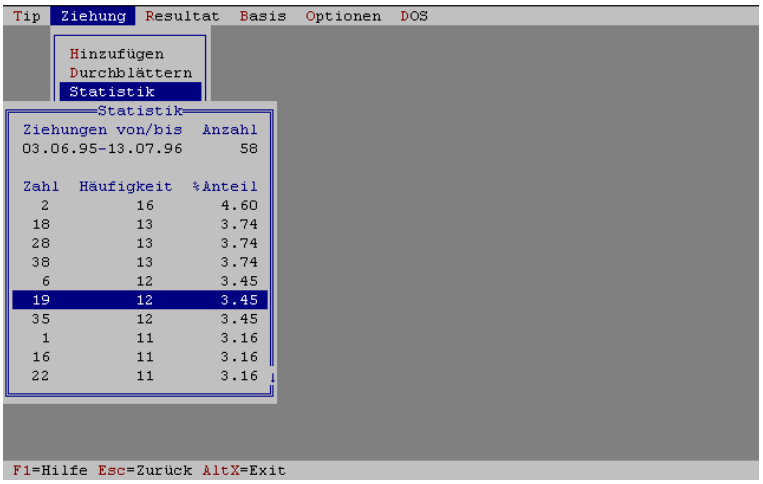
Ziehung



Hier wird das Ergebnis einer Ziehung erfasst.

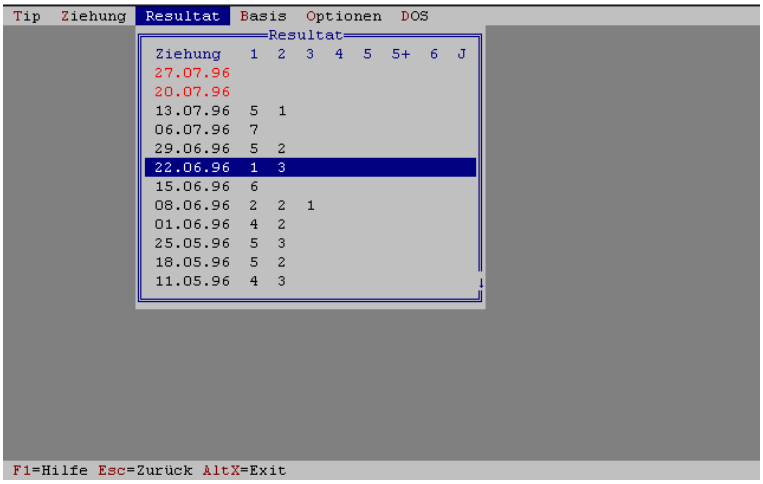
Ziehungen lassen sich durchblättern. [F2] wechselt zum Mutationsbild und [F3] löscht die Zeile.





Auf dem Statistik-Bild sind Häufigkeit und prozentuale Verteilung der Zahlen 1..45, bezogen auf die gespeicherten Ziehungen, ausgewertet.

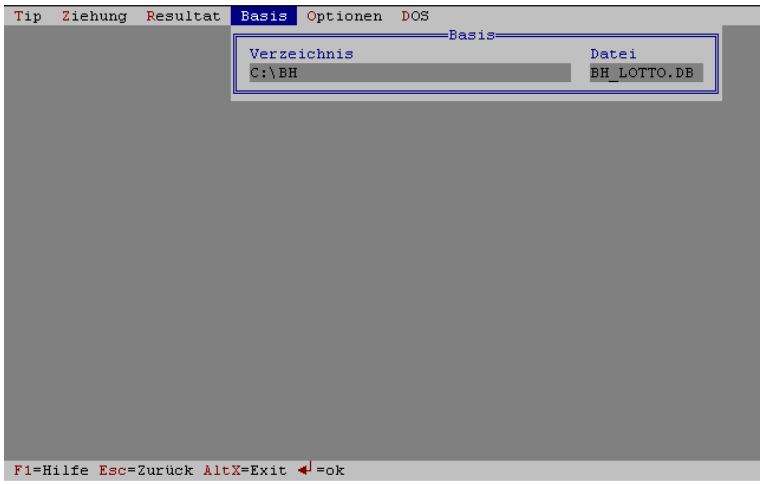
Ergebnis



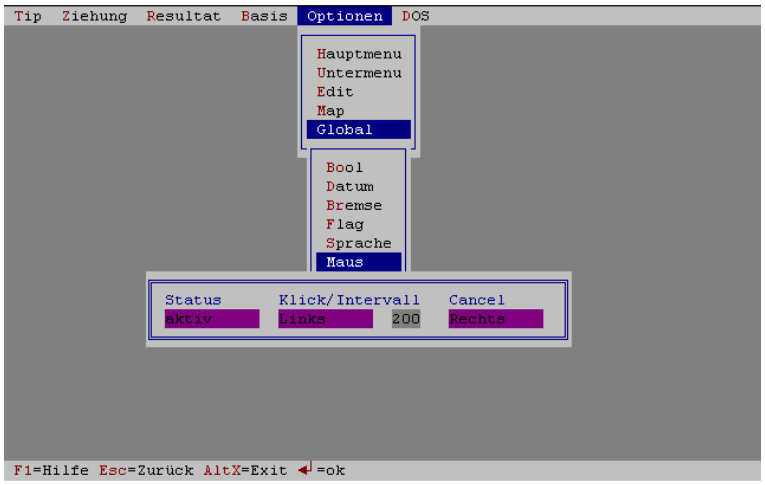
Hier werden die Ergebnisse ausgewertet. Tips ohne Ziehungen sind farblich hervorgehoben.

Basis (Datenbank)

Hier wird die aktive Lotto-Datenbank bestimmt.

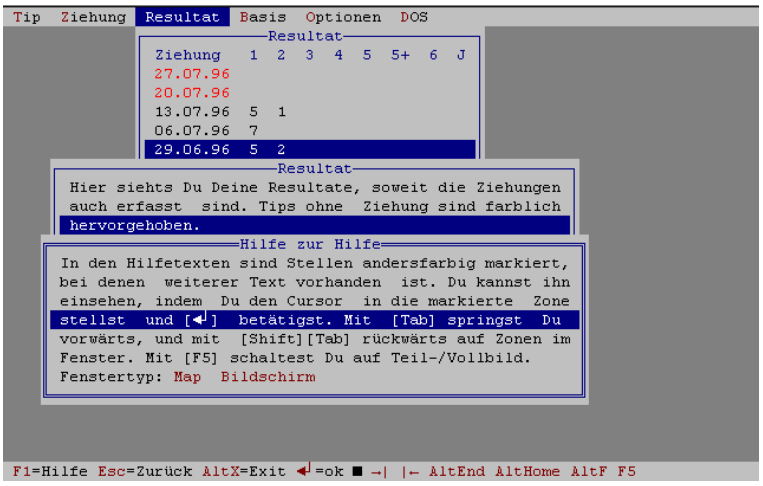


Optionen



Über den Menupunkt «Optionen» lassen sich verschiedene Kriterien den persönlichen Bedürfnissen anpassen. Diese gelten automatisch auch für alle weiteren Produkte der Familie.

Hilfe



Das Hilfesystem ist sensitiv gestaltet und reagiert auf Fehlermeldungen und Cursorposition. Der Inhalt ist nach dem Hypertext-Prinzip aufgebaut. Die Fenster lassen sich vergrössern, verkleinern, verschieben und in den Vordergrund bringen.

Beispielprogramm «Sequenzabstraktion»

SEQ1: Demoprogramm

```
MODULE Seq1;
IMPORT Com_1, Dat_1, FIO, FIOR, FIO_1, Str;
FROM Com_1 IMPORT MaxLR, MinLR, PicRPQ, tLine, tLongReal, tPath;
FROM Dat_1 IMPORT rDate;
FROM Seq_1 IMPORT cSequence;                                    ♣ Seq_1
(*
   Mischen von Datei1 mit Datei2, deren Zeilenaufbau identisch sind
   (0..HIGH(tLine) = '*xxxxxxxx dd.mm.yyyy 9'999'999.99- .....'),
   wobei:
     Zeilen mit '*' in Line[0] überlesen werden,
     Key1 in Line[1..8] aufsteigend,
     Key2 in Line[10..19] absteigend, und
     Key3 in Line[21..33] aufsteigend sortiert sein müssen.
   Die Sequenzen werden überprüft.

   Dieser Modul stellt die Resultate im TEMP-Pfad in 2 Dateien:
   (a) Mischdatei: SEQ1*.$M
   (b) Protokoll:  SEQ1*.$P
*)
TYPE
  tDate = ARRAY[1..10]OF CHAR;
  tKey1 = ARRAY[1..8]OF CHAR;
  tKey3 = LONGREAL;
VAR
  Data1F : FIO.File;
  Data1L : tLine;
  Data2F : FIO.File;
  Data2L : tLine;
  Date : tDate;
  FullPath : tPath;
  Key1 : tKey1;
  Key2 : rDate;
  Key3 : tKey3;
  Line : tLine;
  LongReal : tLongReal;
  MergeF : FIO.File;
  ProtocolF : FIO.File;
  Rd,Wk: LONGCARD;
  Sequ : cSequence;
  i : CARDINAL;
  ok : BOOLEAN;
(*┌──────────────────────────────────────────────────────────────────────┐
  │ Begin Group                                                          │
  └──────────────────────────────────────────────────────────────────────┘*)
PROCEDURE BeginGroup1(Key:ADDRESS);
BEGIN
  Str.Concat(Line,'BeginGroup1 => ',tKey1(Key^));
  FIO.WrStr(ProtocolF,Line); FIO.WrLn(ProtocolF);
END BeginGroup1;

PROCEDURE BeginGroup2(Key:ADDRESS);
VAR
BEGIN
  Dat_1.DateChar(rDate(Key^),'YYYY.MM.DD',Date);
  Str.Concat(Line,'    BeginGroup2 => ',Date);
  FIO.WrStr(ProtocolF,Line); FIO.WrLn(ProtocolF);
END BeginGroup2;

PROCEDURE BeginGroup3(Key:ADDRESS);
VAR
  key : POINTER TO tKey3;
BEGIN
  key:=Key;
  Com_1.LRChr(key^,-7.2,PicRPQ,0,LongReal,ok);
  Str.Concat(Line,'        BeginGroup3 => ',LongReal);
```

```
    FIO.WrStr(ProtocolF,Line); FIO.WrLn(ProtocolF);
  END BeginGroup3;
(*┌──────────────────────────────────────────────────────────────────────┐
  │ Compare Group                                                        │
  └──────────────────────────────────────────────────────────────────────┘*)
  PROCEDURE CompareGroup1(key,Key:ADDRESS):INTEGER;
  BEGIN
    RETURN(Str.Compare(tKey1(key^),tKey1(Key^)));
  END CompareGroup1;

  PROCEDURE CompareGroup2(key,Key:ADDRESS):INTEGER;
  BEGIN
    RETURN(Dat_1.Compare(rDate(key^),rDate(Key^)));
  END CompareGroup2;

  PROCEDURE CompareGroup3(key,Key:ADDRESS):INTEGER;
  VAR
    k,K : POINTER TO tKey3;
  BEGIN
    k:=key; K:=Key;
    IF (k^ < K^) THEN
      RETURN(-1);
    ELSIF (k^ > K^) THEN
      RETURN(+1);
    ELSE
      RETURN(0);
    END;
  END CompareGroup3;
(*┌──────────────────────────────────────────────────────────────────────┐
  │ End Group                                                            │
  └──────────────────────────────────────────────────────────────────────┘*)
  PROCEDURE EndGroup3(Key:ADDRESS);
  VAR
    key : POINTER TO tKey3;
  BEGIN
    key:=Key;
    Com_1.LRChr(key^,-7.2,PicRPQ,0,LongReal,ok);
    Str.Concat(Line,'        EndGroup3   => ',LongReal);
    FIO.WrStr(ProtocolF,Line); FIO.WrLn(ProtocolF);
  END EndGroup3;

  PROCEDURE EndGroup2(Key:ADDRESS);
  BEGIN
    Dat_1.DateChar(rDate(Key^),'YYYY.MM.DD',Date);
    Str.Concat(Line,'    EndGroup2   => ',Date);
    FIO.WrStr(ProtocolF,Line); FIO.WrLn(ProtocolF);
  END EndGroup2;

  PROCEDURE EndGroup1(Key:ADDRESS);
  BEGIN
    Str.Concat(Line,'EndGroup1   => ',tKey1(Key^));
    FIO.WrStr(ProtocolF,Line); FIO.WrLn(ProtocolF);
  END EndGroup1;
(*┌──────────────────────────────────────────────────────────────────────┐
  │ Make Key                                                             │
  └──────────────────────────────────────────────────────────────────────┘*)
  PROCEDURE MakeKeyData1Group1():ADDRESS;
  BEGIN
    RETURN(ADR(Data1L[1]));
  END MakeKeyData1Group1;

  PROCEDURE MakeKeyData1Group2():ADDRESS;
  BEGIN
    Key2:=Dat_1.CharDate(tDate(ADR(Data1L[10])^),'DD.MM.YYYY');
    RETURN(ADR(Key2));
  END MakeKeyData1Group2;

  PROCEDURE MakeKeyData1Group3():ADDRESS;
  BEGIN
    Key3:=Com_1.ChrLR(tLongReal(ADR(Data1L[21])^),-7.2,MinLR,MaxLR,ok);
    RETURN(ADR(Key3));
```

```
  END MakeKeyData1Group3;

  PROCEDURE MakeKeyData2Group1():ADDRESS;
  BEGIN
    RETURN(ADR(Data2L[1]));
  END MakeKeyData2Group1;

  PROCEDURE MakeKeyData2Group2():ADDRESS;
  BEGIN
    Key2:=Dat_1.CharDate(tDate(ADR(Data2L[10])^),'DD.MM.YYYY');
    RETURN(ADR(Key2));
  END MakeKeyData2Group2;

  PROCEDURE MakeKeyData2Group3():ADDRESS;
  BEGIN
    Key3:=Com_1.ChrLR(tLongReal(ADR(Data2L[21])^),-7.2,MinLR,MaxLR,ok);
    RETURN(ADR(Key3));
  END MakeKeyData2Group3;
(*┌──────────────────────────────────────────────────────────────────────┐
  │ Open Data                                                            │
  └──────────────────────────────────────────────────────────────────────┘*)
  PROCEDURE OpenData1():BOOLEAN;
  BEGIN
    FIOR.MakePath(Data1L,'','SEQ1.D1');
    CASE FIO_1.FindFile(Data1L) OF
    | TRUE  : Data1F:=FIO.Open(Data1L);
             RETURN(TRUE);
    | FALSE : RETURN(FALSE);
    END;
  END OpenData1;

  PROCEDURE OpenData2():BOOLEAN;
  BEGIN
    FIOR.MakePath(Data2L,'','SEQ1.D2');
    CASE FIO_1.FindFile(Data2L) OF
    | TRUE  : Data2F:=FIO.Open(Data2L);
             RETURN(TRUE);
    | FALSE : RETURN(FALSE);
    END;
  END OpenData2;
(*┌──────────────────────────────────────────────────────────────────────┐
  │ Read Data                                                            │
  └──────────────────────────────────────────────────────────────────────┘*)
  PROCEDURE ReadData1(VAR Over:BOOLEAN):BOOLEAN;
  BEGIN
    FIO.RdStr(Data1F,Data1L);
    CASE (*%T _mthread *) FIO.ThreadEOF() (*%E *)
         (*%F _mthread *) FIO.EOF (*%E *) OF
    | TRUE  : RETURN(FALSE);
    | FALSE : Over:=(Data1L[0] = '*');
              Com_1.Expand8(Data1L);
              RETURN(TRUE);
    END;
  END ReadData1;

  PROCEDURE ReadData2(VAR Over:BOOLEAN):BOOLEAN;
  BEGIN
    FIO.RdStr(Data2F,Data2L);
    CASE (*%T _mthread *) FIO.ThreadEOF() (*%E *)
         (*%F _mthread *) FIO.EOF (*%E *) OF
    | TRUE  : RETURN(FALSE);
    | FALSE : Over:=(Data2L[0] = '*');
              Com_1.Expand8(Data2L);
              RETURN(TRUE);
    END;
  END ReadData2;
(*┌──────────────────────────────────────────────────────────────────────┐
  │ Work Data                                                            │
  └──────────────────────────────────────────────────────────────────────┘*)
  PROCEDURE WorkData1;
  BEGIN
```

```
    FIO.WrStr(MergeF,Data1L); FIO.WrLn(MergeF);
  END WorkData1;

  PROCEDURE WorkData2;
  BEGIN
    FIO.WrStr(MergeF,Data2L); FIO.WrLn(MergeF);
  END WorkData2;
(*
   Body SEQ1
*)
  BEGIN
    FullPath:=FIO_1.BuildTemp('SEQ','$M');
    MergeF:=FIO.Create(FullPath);
    FullPath:=FIO_1.BuildTemp('SEQ','$P');
    ProtocolF:=FIO.Create(FullPath);
    WITH Sequ DO
      AddGroup(1,CompareGroup1,SIZE(Key1),'>',BeginGroup1,EndGroup1);
      AddGroup(2,CompareGroup2,SIZE(Key2),'<',BeginGroup2,EndGroup2);
      AddGroup(3,CompareGroup3,SIZE(Key3),'>',BeginGroup3,EndGroup3);
      AddData(1,OpenData1,ReadData1,WorkData1,TRUE);
      AddDataGroup(1,1,MakeKeyData1Group1);
      AddDataGroup(1,2,MakeKeyData1Group2);
      AddDataGroup(1,3,MakeKeyData1Group3);
      AddData(2,OpenData2,ReadData2,WorkData2,TRUE);
      AddDataGroup(2,1,MakeKeyData2Group1);
      AddDataGroup(2,2,MakeKeyData2Group2);
      AddDataGroup(2,3,MakeKeyData2Group3);
      ok:=Apply();
      FIO.WrLn(ProtocolF);
      FOR i:=1 TO 2 DO
        ok:=GetCount(SHORTCARD(i),Rd,Wk);
        FIO.WrStr(ProtocolF,'Total Read'); FIO.WrCard(ProtocolF,i,1);
        FIO.WrStr(ProtocolF,'='); FIO.WrLngInt(ProtocolF,Rd,5);
        FIO.WrStr(ProtocolF,'     ');
        FIO.WrStr(ProtocolF,'Total Work'); FIO.WrCard(ProtocolF,i,1);
        FIO.WrStr(ProtocolF,'='); FIO.WrLngInt(ProtocolF,Wk,5);
        FIO.WrLn(ProtocolF);
      END;
      Dispose;
    END;
    FIO.Close(MergeF);
    FIO.Close(ProtocolF);
  END Seq1.
```

Datei SEQ1.D1

```
* Demo-Datei1 zu SEQ1.MOD
*
*    Zeilen mit '*' in Line[0] überlesen,
*    Key1 in Line[1..8] aufsteigend,
*    Key2 in Line[10..19] absteigend, und
*    Key3 in Line[21..33] aufsteigend sortiert.
*    Die Sequenzen werden überprüft.
*
 Albert     1.2.1996    700.50    D1.1
 Albert     1.2.1996    800.50    D1.2
 Albert   11.12.1995    100.25-   D1.3
 Ferdi      1.2.1996    700.50-   D1.4
 Ferdi      1.2.1996    700.50    D1.5
 Franz    15.10.1994  1'255.35    D1.6
 Franz    15.10.1994  1'255.35    D1.7
```

Datei SEQ1.D2

```
* Demo-Datei2 zu SEQ1.MOD
*
*    Zeilen mit '*' in Line[0] überlesen,
*    Key1 in Line[1..8] aufsteigend,
*    Key2 in Line[10..19] absteigend, und
*    Key3 in Line[21..33] aufsteigend sortiert.
*    Die Sequenzen werden überprüft.
*
 Albert     1.2.1996    700.50    D2.1
 Albert     1.2.1996    800.50    D2.2
 Albert   10.11.1995    133.95    D2.3
 Ferdi      1.2.1996    700.50-   D2.4
 Ferdi      1.2.1996    700.50    D2.5
 Franz    16.10.1994  1'255.35    D2.6
 Franz    15.10.1994  1'255.35    D2.7
```

Datei SEQ1.$M (Ergebnis)

```
 Albert     1.2.1996    700.50    D1.1
 Albert     1.2.1996    700.50    D2.1
 Albert     1.2.1996    800.50    D1.2
 Albert     1.2.1996    800.50    D2.2
 Albert   11.12.1995    100.25-   D1.3
 Albert   10.11.1995    133.95    D2.3
 Ferdi      1.2.1996    700.50-   D1.4
 Ferdi      1.2.1996    700.50-   D2.4
 Ferdi      1.2.1996    700.50    D1.5
 Ferdi      1.2.1996    700.50    D2.5
 Franz    16.10.1994  1'255.35    D2.6
 Franz    15.10.1994  1'255.35    D1.6
 Franz    15.10.1994  1'255.35    D1.7
 Franz    15.10.1994  1'255.35    D2.7
```

Datei SEQ1.$P (Protokoll)

```
BeginGroup1 => Albert
    BeginGroup2 => 1996.02.01
        BeginGroup3 =>       700.50
        EndGroup3   =>       700.50
        BeginGroup3 =>       800.50
        EndGroup3   =>       800.50
    EndGroup2   => 1996.02.01
    BeginGroup2 => 1995.12.11
        BeginGroup3 =>       100.25-
        EndGroup3   =>       100.25-
    EndGroup2   => 1995.12.11
    BeginGroup2 => 1995.11.10
        BeginGroup3 =>       133.95
        EndGroup3   =>       133.95
```

```
    EndGroup2   => 1995.11.10
EndGroup1   => Albert
BeginGroup1 => Ferdi
    BeginGroup2 => 1996.02.01
        BeginGroup3 =>       700.50-
        EndGroup3   =>       700.50-
        BeginGroup3 =>       700.50
        EndGroup3   =>       700.50
    EndGroup2   => 1996.02.01
EndGroup1   => Ferdi
BeginGroup1 => Franz
    BeginGroup2 => 1994.10.16
        BeginGroup3 =>     1'255.35
        EndGroup3   =>     1'255.35
    EndGroup2   => 1994.10.16
    BeginGroup2 => 1994.10.15
        BeginGroup3 =>     1'255.35
        EndGroup3   =>     1'255.35
    EndGroup2   => 1994.10.15
EndGroup1   => Franz

Total Read1=   17     Total Work1=    7
Total Read2=   17     Total Work2=    7
```