Keywords fuer Programmstruktur

Neues Programm beginnen	MODULE	Beginnt ein neues Modul. Nachher folgt der Name des Moduls. Am Ende des Moduls steht ein Punkt nach END Modulname.	MODULE Beispiel1; ..... END Beispiel1.
Hilfsprogram-me benutzen	IMPORT	Lädt andere Module, die benützt werden.	IMPORT In, Out, Rn:=RandomNumbers;
Konstanten definieren	CONST	Konstanten sind Variablen, deren Wert im ganzen Modul nicht mehr geändert wird.	CONST maxSize=4;
Variablen-definition	VAR	Keyword, um Variablen zu deklarieren. Wenn ausserhalb der Prozedur -> globale Variable, innerhalb einer Prozedur -> lokale Variable.	VAR r:REAL; c: CHAR;
Prozedur	PROCEDURE	Beginnt eine neues Unterprogramm/ Subroutine. Die Signatur einer Subroutine besteht aus „PROCEDURE“, dem Namen der Subroutine, ev. einem „*“, um die Subroutine zu exportieren, den mitgegebenen Parametern und dem Typ der Rückgabe.	PROCEDURE Test*(laenge, breite: INTEGER): INTEGER;
Prozedur beginnen	BEGIN	Start der eigentlichen Befehlssequenz innerhalb einer Subroutine (oder auch des Moduls).	BEGIN
Prozedur beenden	END	Schluss eines Unterprogramms, gefolgt vom Namen des Unterprogramms.	END Test;
Kommentar	(* *)	Mit diesen Klammern werden Kommentare eingefügt. Jedes Programm braucht Kommentare, um die Lesbarkeit zu erhöhen.	(* Auf dieser Zeile passiert folgendes: ... *)

Ablauf zum Erstellen eines Programms:

- Edit.Open Dummy.Mod hinschreiben, mit Mitteltaste anklicken -> Oeffnet viewer mit neuem File.
- Modulskelett eingeben (MODULE, IMPORT, TYPE, ev. VAR und PROCEDURE-Signaturen. Die Prozedur, mit der du das Programm starten willst, muss exportiert werden, d.h. muss einen Stern nach dem Namen haben: PROCEDURE Test*();
- Compilieren mit Compiler.Compile
- Syntaxfehler anhand der Fehlercodes eliminieren :-)
- Falls dann etwas an Modulkopf oder Signaturen geandert wurde, mit Compiler.Compile *\s compilieren, erstellt neues Symbolfile
- Beginne nun mit dem eigentlichen Ausprogrammieren.
- Um das Programm zu starten, schreibst du unter das Programm die Zeile Modulname.Prozedurname~ . Die Wellenlinie ist dazu da, dem Computer zu signalisieren, dass der Befehl dort fertig ist. Nun kannst du es mit einem Mittelklick auf diese Zeile starten.
- Wenn du eine neue Version compiliert hast, musst du zuerst die alte aus dem Speicher entfernen, damit du die neuen Ergebnisse sehen kannst: System.Free Modulname~

So koennte ein Modulskelett aussehen:

```
MODULE Versuch1;
      IMPORT Out;
      PROCEDURE ausfuehren*();
            BEGIN
      END ausfuehren;
END Versuch1.

Versuch1.ausfuehren~
System.Free Versuch1~
```

Variablentypen und -manipulationen

brinerth@iiic.ethz.ch

Oberon Keywords - Info 1

Zahlvariablen-typen	INTEGER, REAL	INTEGER ist der Ganzzahltyp, REAL sind Zahlen mit Kommastellen	VAR i:INTEGER; r:REAL; i:=3; r:=2.4736;
Zeichen-variablen	CHAR	Es wird ein einzelnes Zeichen in die CHAR Variable gespeichert.	VAR c:CHAR; c:='a';
Boolsche Variablen	BOOLEAN	Variablen, die nur 2 mögliche Werte annehmen können: wahr oder falsch	VAR b:BOOLEAN; b:=TRUE;
Zuweisung	:=	Ein Wert wird einer vorher deklarierten Variable zugewiesen.	i:=2; r:=3.14159268; c:=FALSE;
Vergleichs-operatoren	#, =, <=, >=, >, <	Numerische Werte können verglichen werden. Boolsche Werte können nur auf = oder # verglichen werden. Das Ergebnis eines Vergleichs ist immer ein boolscher Wert, nämlich wahr oder falsch.	b:=(x >= y);
Zahloperatoren	+, -, *, /, DIV, MOD	Die üblichen Zahloperatoren, können aber nur auf Zahlvariablen angewendet werden. DIV ist die Ganzzahldivision. MOD ist die Restberechnung.	17 DIV 3 -> gibt 5, 17 MOD 3 -> gibt 2
Boolsche Operatoren	&, OR, ~	Operatoren für „and“, „or“ und „not“	b:= b1 & (~b2);
Ausgabe	Out.Int Out.String	Benützt Subroutinen aus dem Modul Out (-> muss bei IMPORT eingebunden werden), um Ausgaben auf den Bildschirm zu machen. 2. Parameter bei Out.Int: Anzahl Stellen	Out.Int(x,7); Out.String(„Hallo“); Out.Char(d); Out.Real(r,20);
Eingabe	In.Open(), In.Int(x)	Die Eingabe in Oberon funktioniert ziemlich speziell. Sie ist nicht interaktiv, es geht also nicht irgendein Fenster auf, sondern die Eingabedaten werden direkt hinter dem Programmaufruf hingeschrieben. Mit dem Befehl „In.Open“ wird die Eingabe eröffnet und der Lesezeiger direkt hinter den Prgaufruf gesetzt. Nun kann mit In.Int(x) etc immer eine Einheit des bestimmten Typs eingelesen werden.	In.Open(); In.Int(x); In.Char(c); In.Real(r); Programmaufruf zB: Versuch1.ausfuehren 345 f 4.1446~
Rückgabe aus Prozedur	RETURN	Um einen Wert aus einer Subroutine zurück zu geben, wird die Subroutine mit RETURN beendet und der direkt dahinter stehende Wert zurückgegeben. Der Typ dieses Werts muss mit dem in der Signatur angekündigten Typ übereinstimmen.	RETURN x; oder RETURN (b=c) -> gibt wahr oder falsch zurück

brinerth@iiic.ethz.ch

Statische Datenstrukturen

Array	ARRAY	Ein Gitter, in dem mehrere Werte vom gleichen Typ gespeichert werden können. Sie werden mit dem Variablennamen und einem Index adressiert	VAR a:ARRAY 16 OF INTEGER; a[3]:=6;
Länge eines ARRAYs	LEN(a)	Gibt die Länge (= Anzahl Kästchen) eines Arrays zurück.	VAR a:ARRAY 16 OF INTEGER; i:INTEGER; i:=LEN(a);
Record	RECORD	Eine Sammlung von beliebigen Datentypen mit einem neuen Namen. Die einzelnen Variablen werden adressiert mit dem Recordnamen, dann einem Punkt und dann dem Feldnamen. Speziell: Vererbung: Ein RECORD kann einen anderen RECORD erweitern, d.h. er uebernimmt die Felder des anderen und fuegt noch neue dazu.	TYPE Person = RECORD name:ARRAY 16 OF CHAR; groesse: REAL; END; TYPE Student = RECORD (Person) NR: INTEGER; END;
Menge	SET	Ein Datentyp, der eine Menge darstellt, die die Zahlen 0 bis 31 enthalten kann. Ist sehr effizient und speichersparend, um zB höchstens 32 Elemente „abzuhaken“, ob sie schon vorgekommen sind.	VAR s:SET;

Mengen-operationen	INCL, EXCL, IN	Damit werden Elemente ins Set hineingenommen oder wieder rausgeschmissen. Mit IN wird getestet, ob ein Element drin ist.	INCL(s,i);
			EXCL(s,i);
			IF (i IN s) THEN
			Out.String(„enthalten“);
			END;

Neuer selbst-definierter Typ	TYPE	Neue komplexe Typen können selber definiert werden. Sie können nachher wie die Basistypen benützt werden.	TYPE String = ARRAY 32 OF CHAR; VAR s:String;
CHAR -> INTEGER	ORD(c)	Die Zahl gemäss ASCII-Code, die einem bestimmten Zeichen zugeordnet ist, wird zurückgegeben.	VAR c:CHAR; i:INTEGER;c:='a'; i:=ORD(c);
INTEGER -> CHAR	CHR(x)	Umgekehrte Funktion von ORD(c). Gibt das Zeichen zurück, das einer INTEGER Zahl zugeordnet wird.	VAR c:CHAR; i:INTEGER;i:=97; c:=CHR(i);
Nullzeichen	0x	Das Zeichen mit Zahl 0 des ASCII Codes. Damit wird ein String (=Array von CHAR) abgeschlossen, sofern das ARRAY gross genug für den String ist.	VAR c:CHAR; c:=0x;

brinerth@iiic.ethz.ch

Kontrollflusssteuerung

Bedingung	IF THEN ELSIF ELSE END	Nach dem IF muss ein Ausdruck kommen, der zu TRUE oder FALSE ausgewertet werden kann. Je nachdem wird die Befehlssequenz nach THEN ausgeführt oder nicht. Es können beliebig viele ELSIF folgen (auch keines) und höchstens 1 ELSE. Das END muss immer stehen.	IF (i>0) THEN
			  i:=5;
			ELSIF (i<0) THEN
			  i:=3;
			ELSE
			  i:=7;
			END;

Mehrere Varianten	CASE OF END	Falls eine Variable auf mehrere mögliche Werte abgefragt werden muss, kann anstelle eines IF mit vielen ELSIFs auch das CASE Statement benutzt werden. Falls keine der Varianten zutrifft, kann analog zum IF ein ELSE Statement folgen.	CASE x OF 0:
			  Out.String(„Null“);
			| 1:
			  Out.String(„eins“);
			ELSE
			Out.String(„was anderes“); END;

Absicherung	ASSERT	Hier kann auf eine Bedingung geprüft werden, ob sie TRUE ist. Falls das nicht der Fall ist, bricht das Programm mit einer Fehlermeldung „TRAP ASSERTION failed“ ab.	ASSERT (i>2);
Schleife for	FOR DO END	Die For-Schleife benötigt eine INTEGER Zählvariable, die bei jedem Durchgang erhöht/erniedrigt wird.	FOR i:=1 TO 100 BY 3 DO Out.String(„schon wieder“); END;
Schleife while	WHILE DO	Bei der While-Schleife steht die Bedingung am Anfang. Falls sie schon beim Beginn nicht TRUE ist, wird die Schleife kein einziges Mal durchlaufen.	WHILE i<=10 DO i:=i+1; END;
Schleife until	REPEAT UNTIL	Bei der Repeat-Schleife kommt die Bedingung erst am Schluss. Die Schleife wird also auf jeden Fall mindestens 1 Mal durchlaufen.	REPEAT i:=i+1; UNTIL (i>10);
Schleife loop	LOOP END	Das ist die einzige Schleife, die keine Abbruchbedingung hat. Muss mit EXIT beendet werden.	LOOP i:=i+1; IF i>10 THEN EXIT; END; END;
Aus Schleife rausspringen	EXIT	Wenn dieser Befehl erreicht wird, wird aus der (innersten) Schleife herausgesprungen. Wird vor allem bei LOOP gebraucht.	IF (i>0) THEN EXIT; END;
Notbremse	HALT(99)	So kann geprüft werden, ob ein Programm bis zu einer bestimmten Stelle kommt. Sobald es den Befehl HALT erreicht, bricht das Programm mit TRAP und der mitgegebenen Nummer ab.	HALT(87);
Call by reference	VAR	Das Schlüsselwort VAR in einer Prozedursignatur zeigt an, dass dieser Parameter nicht als Kopie übergeben wird, sondern als Verweis auf das Original und deshalb der Originalwert auch geändert werden kann.	PROCEDURE Test*(VAR laenge: INTEGER);

brinerth@iiic.ethz.ch

Dynamische Datenstrukturen

Referenz auf Objekt	POINTER TO	Ein Pointer ist ein Variablentyp, der auf einen bestimmten Bereich im Speicher zeigt. Mit TYPE wird ein solcher POINTER definiert. Dynamische Datenstrukturen bestehen aus solchen POINTERn. Meist zeigt ein POINTER auf einen RECORD. Im Beispiel recht ist n die Adresse eines Speicherbereichs. n^ heisst „Derefenzierung“ und ist nicht die Adresse, sondern der RECORD nodedesc selber.	TYPE node = POINTER TO nodedesc;
			nodedesc = RECORD
			  key:INTEGER;
			  left,right: node;
			END;
			
			VAR n:node;

neues Element erzeugen	NEW	Damit wird fuer den RECORD, auf den der POINTER zeigt, der Speicherplatz bereitgestellt. Die Schwierigkeit ist immer zu entscheiden, braucht es effektiv einen neuen RECORD, oder nur einen neuen Pfeil, der darauf zeigt.	VAR n:node; NEW(n);
Nullpointer	NIL	Ein Pointer der „nirgendwo“ hinzeigt, heisst ein NIL-Pointer.	IF (node=NIL) THEN Out.String(„NIL-Pointer“); END;
Typenabfrage	IS	Es kann geprüft werden, ob ein RECORD oder POINTER von einem bestimmten Typ ist.	IF (n IS node) THEN Out.String(„ist node“);

Typen-zusicherung	WITH DO END	Erklaert dem Compiler, dass eine bestimmte Variable fuer einen ganzen Bereich garantiert von einem bestimmten Typ ist. Falls das dann nicht stimmen sollte, gibt es einen Laufzeitfehler	VAR n:node;
			WITH n:node DO
			  n^.left:=n;
			END;

Diverses

+1, -1	INC, DEC	Einfach eine Verkürzung für i:=i+1 -> INC(i) und i:=i-1 -> DEC(i)	INC(i); DEC(i);
String kopieren	COPY(s,x)	Mit diesem Befehl können Strings (=ARRAYs OF CHAR) von verschiedener Länge kopiert werden. Der String s wird in den String x kopiert, solange, bis bei s das Terminierungs-zeichen 0x auftritt oder der String zu Ende ist.	COPY(s,x);
Grossbuch-staben	CAP	Macht aus einem CHAR den zugehörigen Grossbuchstaben.	VAR c:CHAR; c:=CAP(c);

auf ungerade testen	ODD	Prüft eine INTEGER Zahl darauf, ob sie ungerade ist und gibt in diesem Fall TRUE zurück.	IF (ODD(i)) THEN
			Out.String(„ungerade“);
			END;

Betrag	ABS	Gibt den Betrag einer Zahl zurück.	j:=ABS(i)
REAL -> INTEGER	ENTIER(r)	Schneidet die Nachkommastellen ab.	VAR r:REAL; li: LONGINT; li:=ENTIER(r);
Zahlvariablen-typen	SHORTINT, LONGINT, LONGREAL	Für die Zahlvariablentypen gibt es verschiedene Längen, um verschieden grosse Zahlenräume abzudecken.	VAR si:SHORTINT; lr: LONGREAL;
Zahl verkürzen	SHORT(i)	Kuerzt immer auf den naechst kuerzeren Typ, zB LONGINT wird zu INTEGER etc	VAR li:LONGINT; i:INTEGER i:=SHORT(li);
Zahl verlängern	LONG(x)	Verlaengert auf den naechstlaengeren Typ	VAR r:REAL; lr:LONGREAL lr:=LONG(r)

grösstmöglicher Wert	MAX(INTEGER)	Bei Zahlvariablen: Gibt groessmoeglichen Wert zurueck.
		Bei SET: Gibt groesste Zahl, die im SET enthalten ist zurueck.
		MIN ist das Gegenstueck dazu.
			VAR i:INTEGER;s:SET;
			i:=MAX(INTEGER);
			i:=MAX(SET);

Speichergrösse eines Typs	SIZE	Gibt zurueck, wieviel Platz im Speicher fuer einen bestimmten Datentyp reserviert ist.	VAR i:INTEGER; i:=SIZE(REAL);

brinerth@iiic.ethz.ch

brinerth@iiic.ethz.ch