MODULA-3

Typen

Standard: `INTEGER`, `CARDINAL`, `REAL`, `LONGREAL`, `EXTENDED`, `CHAR`, `BOOLEAN`, `TEXT`
- Konstanten: Zahlen `xxx`; FPs: `x.xxExxDO`; Zeichen: `'c'`; Texte: `"..."`; Boolean: `TRUE/FALSE`
- Fkt für Ordinaltypen: `INC(x)`, `DEC(x)`, `FIRST(type)`, `LAST(type)`, `ORD(x)`, `VAL(x, type)`, `NUMBER(x)`

Subtypen (von Ordinaltypen): `Type=[first..last]`
- Konstanten: Konstanten der Basistypen im entsprechenden Bereich

Aufzählungstyp: `Type={Ident, ...}`
- Konstanten: Type.Ident ist das angegebenen Element
- Operationen: Zuweisung, Vergleich

Mengentyp: `Type=SET OF Subtype`
- Konstanten: `Type{Ident,...}` ist Menge mit den aufgezählten Elementen (auch `Type{}`)
- Operationen: Zuweisung, Vereinigung `+`, Differenz `-`, Schnitt: `*`, Element: `IN`, Vergleiche

Arrays: `Type=ARRAY Subtype, ... OF Type`
- Konstanten: `Type{elem, ...}` - abgeschl. mit "..": mit letztem angegebenen auffüllen
- Operationen: Zuweisung, Test auf Gleichheit oder Ungleichheit, `FIRST(x)`, `LAST(x)`, `NUMBER(x)`

Records: `Type=RECORD vardecl END;`
- Konstanten: `Type{Value,...}` oder `Type{Ident:=Value, ...}`
- Operationen: Zuweisung, Komponente (`Var.Component`), Gleichheit, Ungleichheit
- With-Anweisung: `WITH binding=variable, ... DO [...] END`

Prozedurtyp: `Type=PROCEDURE (paramlist) [: type];`

Zeiger: `Type=REF Type;`
- Konstanten: `NIL`
- Operationen: Objekt anlegen, Zeiger dereferenzieren, Zuweisung, Test auf Gleichheit/Ungleichheit
- Objekt anlegen: `var:=NEW(Type);` bei Record-Refs auch: `var:=NEW(Type, element:=initexpr, ...);`
- Dereferenzierung: `pointervar^`

Subtypisierung: `Type <: Supertype` (nur auf Zeigertypen erlaubt!!!);
- Auflösung: `REVEAL Type=[BRANDED] REF Type;`
- Subtypenordnung: NULL <: REF T <: (S<:T) <: REFANY

Module und Prozeduren

Module
- `MODULE Module EXPORTS Interface; declarations; functions; BEGIN [...] END Module.`
- `INTERFACE Interface; declarations; functiondecl; BEGIN init END Interface.`
- `IMPORT Interface [AS Name];`
- `FROM Interface IMPORT Function, ...;`

Prozeduren
- `PROCEDURE name (paramlist) [: returntype] [RAISES {exception, ...}] = ...`
- call-by-value: `VALUE param, ...: Type [:=initexpr];` -oder- `param, ...: Type [:=initexpr];`
- call-by-reference: `VAR param, ...: Type;`
- Aufruf: `name(expr, ...);` -oder- `name(param:=expr, ...);`
- "Schlucken" des Rückgabewerts: `EVAL name(expr, ...);`

Deklarationen
- `VAR name, ...: Type [:=initexpr];`
- `CONST name [:Type] = initexpr;`

- `TYPE type=typeconstructor;`
- `EXCEPTION ( name | name(paramtype) ), ...;`
- Funktionsdeklaration: `PROCEDURE name (params) [: type] [RAISES {...}];`
– Deklararierte Funktionen müssen später definiert werden!!!

Kontrollstrukturen

```
IF boolexpr THEN [...] { ELSIF boolexpr THEN [...] } [ ELSE [...] ] END;
CASE ordexpr OF { | value => stmt; ...; } ELSE stmt; ... END; !!! Else immer verwenden !!!
WHILE boolexpr DO [...] END;
REPEAT [...] UNTIL boolexpr;
FOR Ident := start TO end [BY step] DO [...] END; !!! Laufvariable nicht deklarieren !!!
LOOP [...] EXIT; [...] END;
```

Ausnahmebehandlung

Assert-Pragma: `<* ASSERT boolexpr *>`

```
TRY [...] EXCEPT { | exception(variable) => stmt; ...; } [ELSE stmt; ...] END;
TRY [...] FINALLY [...] END;
```

Auslösen einer Exception: `RAISE exception;` -oder- `RAISE exception(value);`

Objektorientierte Programmierung

Klassendefinition

- `TYPE Class = Super OBJECT property; ...; METHODS method;...; OVERRIDES override; ...; END;`
- Eigenschaftendeklaration: `name: Type [:=initexpr];`
- Methodendeklaration: `name(paramlist) [: Type] := function;` (lokale Fkt. ohne Parameterliste)
- virtuelle Methodendeklaration: `name(paramlist) [: Type] := NIL;` (`:=NIL` ist optional)
- Redefinition: `method := function;`
- function muß später definiert werden: `function (self: class; paramlist) [: Type] = ...`
- Suptypisierungsreihenfolge: NULL $<:$ T OBJECT $<:$ T $<:$ ROOT $<:$ REFANY
- Information-Hiding: `TYPE T<:Public; Public = ROOT OBJECT ... END;`
- In der öffentlichen Klasse sollte eine Methode "`init(): T:=NIL`" deklariert sein.

Anlegen von Objekten

- Objektdeklaration: `TYPE object, ...: class;`
- Anlegen: `object := NEW(class);` -oder- `object := NEW(class).init();` (`init()` muß Objekt liefern!)

Wichtige Module

Modul SIO

- `GetChar()`, `PutChar(ch)`, `*Int`, `*Text`, `*Real`, `*LongReal`, `*Bool`
- auch `Get*(Rd.T)` bzw. `Put*(var, Wr.T)`

Modul Text

- `Equal(a,b)`, `GetChar(t, pos)`, `Length(t)`, `Empty(t)`, `FromChar(c)`, `FindChar(t,c,start)`

Modul Rd [definiert Typ `T` und die Exceptions `EndOfFile` sowie `Failure(AtomList.T)`]

- `EOF(T)`, `GetChar(T,num): text`, `GetLine(T): text`, `Close(T)`, `Closed(T): bool`, `Length(T):num`

Modul Wr [definiert Typ `T` und die Exception `Failure(AtomList.T)`]

- `PutChar(T, char)`, `PutText(T, text)`, `Close(T)`, `Closed(T)`, `Length(T)`

Modul SF

- `OpenRead(txt)`, `OpenAppend(txt)`, `OpenWrite(txt)`, `FileExists(txt)`, `CloseRead(Rd.T)`, `CloseWrite(WR.T)`