Loeng 5.
Dünaamilised objektieksemplarid.

Baasnäide:

```
TYPE
        PunktiViidaTyyp = ^Punkt;
        RingiViidaTyyp = ^Ring;
        KolmnurgaViidaTyyp = ^Kolmnurk;
. . .
VAR
        PunktiViit: PunktiViidaTyyp;
        RingiViit: RingiViidaTyyp;
        KolmnurgaViit: ^Kolmnurk;
```

- **Mälu eraldamine dünaamilisele eksemplarile**:
  a) New (viitmuutuja);
     Näide: New (KolmnurgaViit);
     ```
     KolmnurgaViit^.Init (250, 150, Blue, 50);
     . . .
     KolmnurgaViit^.Tiiruta (100);
     ```
  b) New (viitmuutuja, konstruktoriväljakutse);
     Näide: New (KolmnurgaViit, Init (250, 150, Blue, 50));
     NB! Tegeliku mälueralduse teostab siin konstruktor!
  c) viitmuutuja := New (viidatüüp);
     Näide: KolmurgaViit := New (KolmnurgaViidaTyyp);
  d) viitmuutuja := New (viidatüüp, konstruktoriväljakutse);
     Näide:
     PunktiViit := New (PunktiViidaTyyp, Init (150, 100, Red));
     või
     PunktiViit := New (KolmnurgaViidaTyyp, Init (0, 2, Red, 50));
     . . .
     Milles on erinevus?

- **Dünaamilisele eksemplarile eraldatud mälu vabastamine**:
  a) Dispose (viitmuutuja);
     Näide: Dispose (PunktiViit);
     Milline probleem võib siin tekkida?

     Destruktor on protseduur, mille peamiseks funktsiooniks on ligipääsu võimaldamine eksemplari VMT-le ja selle kaudu eksemplari mälupiirkonna mõõdule, mida on vaja teada vabastatava mälu mahu kindlakstegemiseks.
     Destruktori tüüpkuju: DESTRUCTOR Done; VIRTUAL;

     Näite õige variant: PunktiViit^.Done;
     Dispose (PunktiViit);

  b) Dispose (viitmuutuja, destruktoriväljakutse);
     Näide: Dispose (PunktiViit, Done);

Dünaamilised objektistruktuurid ja polümorfism.

Joonis 5.1. Lausepuhvri skeem.

Näide 5.1. Moodul, mis realiseerib lausepuhvri.

```
UNIT Laused;

INTERFACE

USES Crt;
TYPE
    Lause = OBJECT
        Tekst: String;
        CONSTRUCTOR Init (SeeTekst: String);
        DESTRUCTOR Done; VIRTUAL;
        PROCEDURE Kuva; VIRTUAL;
        FUNCTION LeiaTekst: String;
    END;

    PositsLause = OBJECT (Lause)
        X, Y: Byte;
        CONSTRUCTOR Init (SeeTekst: String;
                          AlgX, AlgY: Byte);
        PROCEDURE Kuva; VIRTUAL;
        FUNCTION LeiaX: Byte;
        FUNCTION LeiaY: Byte;
    END;

    VarvilineLause = OBJECT (PositsLause)
        Varv: Byte;
        CONSTRUCTOR Init (SeeTekst: String;
                          AlgX, AlgY, AlgVarv: Byte);
        PROCEDURE Kuva; VIRTUAL;
        FUNCTION LeiaVarv: Byte;
    END;
```

```
    LauseViit = ^Lause;
    PositsLauseViit = ^PositsLause;
    VarviliseLauseViit = ^VarvilineLause;

    SolmeViit = ^Solm;
    Solm = RECORD
        Info: LauseViit;
        J_ne: SolmeViit;
    END;

    Puhver = OBJECT
        Solmed: SolmeViit;
        CONSTRUCTOR Init;
        DESTRUCTOR Done; VIRTUAL;
        PROCEDURE LisaLause (UusLause: LauseViit);
        PROCEDURE KuvaLaused;
    END;


IMPLEMENTATION

    CONSTRUCTOR Lause.Init (SeeTekst: String);
    BEGIN
        Tekst := SeeTekst;
    END;

    DESTRUCTOR Lause.Done;
    BEGIN
    END;

    PROCEDURE Lause.Kuva;
    BEGIN
        Writeln (Tekst);
    END;
```

```
FUNCTION Lause.LeiaTekst: String;
BEGIN
	LeiaTekst := Tekst;
END;

CONSTRUCTOR PositsLause.Init (SeeTekst: String;
                              AlgX, AlgY: Byte);
BEGIN
	Lause.Init (SeeTekst);
	X := AlgX;
	Y := AlgY;
END;

PROCEDURE PositsLause.Kuva;
BEGIN
	GotoXY (X, Y);
	Lause.Kuva;
END;

FUNCTION PositsLause.LeiaX: Byte;
BEGIN
	LeiaX := X;
END;

FUNCTION PositsLause.LeiaY: Byte;
BEGIN
	LeiaY := Y;
END;

CONSTRUCTOR VarvilineLause.Init (SeeTekst: String;
                                 AlgX, AlgY: Byte;
                                 AlgVarv: Byte);
BEGIN
	PositsLause.Init (SeeTekst, AlgX, AlgY);
	Varv := AlgVarv;
END;
PROCEDURE VarvilineLause.Kuva;
```

```
        VAR EndineVarv: Byte;
BEGIN
        EndineVarv := TextAttr;
        TextAttr := Varv;
        PositsLause.Kuva;
        TextAttr := EndineVarv;
END;

FUNCTION VarvilineLause.LeiaVarv: Byte;
BEGIN
        LeiaVarv := Varv;
END;

CONSTRUCTOR Puhver.Init;
BEGIN
        Solmed := NIL;
END;

DESTRUCTOR Puhver.Done;
        VAR Osuti: SolmeViit;
BEGIN
        WHILE Solmed <> NIL DO
                BEGIN
                        Osuti := Solmed;
                        Dispose (Osuti^.Info, Done);
                        Solmed := Osuti^.J_ne;
                        Dispose (Osuti);
                END;
END;
```

```
    PROCEDURE Puhver.LisaLause (UusLause: LauseViit);
        VAR Osuti: SolmeViit;
    BEGIN
        New (Osuti);
        Osuti^.Info := UusLause;
        Osuti^.J_ne := Solmed;
        Solmed := Osuti;
    END;

    PROCEDURE Puhver.KuvaLaused;
        VAR Osuti: SolmeViit;
    BEGIN
        Osuti := Solmed;
        WHILE Osuti <> NIL DO
            BEGIN
                Osuti^.Info^.Kuva;          { polümorfism! }
                Osuti := Osuti^.J_ne;
            END;
    END;
END;
```

Näide 5.2. Programm, mis kasutab moodulit “Laused”.

Geneeriline moodul.

- **Geneerilise mooduli loomise põhiidee**: koostame mooduli, milles operatsioonid dünaamiliste andmestruktuuridega on realiseeritud fiktiivse, andmeid mittesisaldava objekti jaoks ja jätame konkreetse andmetüübi selle objekti järglaste otsustada.

Joonis 5.2. Järjekorra skeem.

Joonis 5.3. Geneeriliste järjekorraoperatsioonide realiseerimise skeem.

Näide 5.3. Geneeriline moodul järjekorraoperatsioonide jaoks.