1.

a. Hur skulle man skriva

```
WHILE villkor DO
  satser
END
```

om man istället för `WHILE`-satsen var tvungen att använda en `LOOP`-sats?

(3 poäng)

b. `INTEGER` och `CHAR` är exempel på skalära typer. Skalära typer kan t ex användas som indextyp i arraytyper. Ge två andra exempel på skalära typer och två andra exempel på programkonstruktioner där skalära typer (men inte andra typer) kan användas.

(4 poäng)

2.

a. Skriv en procedur `ReadBool`,

```
PROCEDURE ReadBool(VAR b:BOOLEAN; VAR success:BOOLEAN);
```

som läser in ett sanningsvärde. Parametern `b` ska sättas till `TRUE` resp `FALSE` om man kan läsa in ordet `TRUE` resp `FALSE`. Parametern `success` ska sättas till `TRUE` om inläsningen lyckades och annars till `FALSE`. *(5 poäng)*

b. Hur många `*`:or skrivs ut då följande program körs? Motivera svaret.

```
MODULE MångaStjärnorBlirDet;

FROM InOut IMPORT Write,WriteLn;

PROCEDURE p(n:CARDINAL);
VAR i:CARDINAL;
BEGIN
  FOR i:=1 TO 2*n DO
    Write("*")
  END;
  WriteLn
END p;

VAR i:CARDINAL;

BEGIN
  FOR i:=1 TO 30 DO
    p(i)
  END
END MångaStjärnorBlirDet.
```

(4 poäng)

3.

a. Antag att du har stött på ett Modula-2-system vars `String`-modul bara innehåller procedurerna `Length` och `Assign`. Gör en implementering av proceduren `Concat` som kan användas i detta Modula-2-system. (För beskrivningar av de nämnda procedurerna, se bilagan med biblioteksmoduler.) *(8 päng)*

b. I vissa Modula-2-system kan inte strukturerade typer, utan bara skalära typer och pekare, vara resultat från funktioner. Skriv om programfragmentet nedan så att det kan användas i ett sådant Modula-2-system. *(5 poäng)*

```
TYPE Point = RECORD x,y:REAL END;

PROCEDURE add(p1,p2:Point):Point;
VAR p:Point;
BEGIN
  p.x:=p1.x+p2.x;
  p.y:=p1.y+p2.y;
  RETURN p
END add;
.
.
.
  p4:=add(add(p1,p2),p3);
```

4. Skriv en komplett implementeringsdel till följande modul:

```
DEFINITION MODULE BokListModul;
(* En modul för länkade listor av böcker. *)

FROM BokModul IMPORT Bok; (* Bok är en typ för böcker *)

TYPE BokLista = POINTER TO BokNod; (* borde kanske vara abstarkt *)
     BokNod = RECORD
                head:Bok;
                tail:BokLista
              END;

CONST Nil = NIL; (* Nil är en tom boklista *)

PROCEDURE cons(b:Bok; bs:BokLista):BokLista;
(* cons lägger till boken b först i listan bs. *)

PROCEDURE null(bs:BokLista):BOOLEAN;
(* null ger TRUE om listan bs är tom, annars FALSE *)

PROCEDURE hd(bs:BokLista):Bok;
(* hd ger den första boken i listan bs.
   bs får inte vara tom *)

END BokListModul.
```

(10 poäng)

5. Skriv en funktionsprocedur som för ett givet bokregister ger en lista med alla böcker av en viss författare. Följande typer används för böcker och bokregister:

```
TYPE Namn  = ARRAY [1..20] OF CHAR;
     Titel = ARRAY [1..50] OF CHAR;

     Bok = RECORD
             forfattare,forlag:Namn;
             titel:Titel;
             sidor:CARDINAL;
           END;

CONST max=10000;

TYPE BokRegister = RECORD
                     bok:ARRAY [1..max] OF Bok;
                         (* böckerna är inte sorterade *)
                     antal: [0..max] (* antal böcker i registret *)
                   END;
```

Använd modulen för listor av böcker från uppgift 4. Din funktionsprocedur ska ha följande huvud:

```
PROCEDURE allaBockerAv(forf:Namn; r:BokRegister):BokLista;
```

(8 poäng)

6. Antag att man i ett visst program behöver hantera reell-värda funktioner av en reell variabel, där samtliga funktioner kan skrivas på följande form:

$$A\cos x + B\sin x + k_0 + k_1 x + \ldots + k_n x^n$$

(Man kan anta att $n = 10$ till att börja med, men detta kan behöva ändras.)

Det kan vara lämpligt göra en separat modul för detta. Vi kan kalla modulen `FunktionsModul`. Antag att modulen behöver tillhandahålla följande:

- En typ, kalla den `Funktion`, för funktioner av ovan beskrivna form.
- En funktionsprocedur `addera` för att addera två funktioner
- En funktionsprocedur `derivera` för att derivera en funktion.
- En funktionsprocedur `eval` för att beräkna värdet av en funktion för ett givet värde på variabeln.

a. Skriv definitionsdelen till modulen `FunktionsModul`. Typen `Funktion` behöver inte vara abstrakt. *(5poäng)*

b. Skriv en komplett implementeringsdel till modulen `FunktionsModul`.

(8 poäng)

Biblioteksmoduler

```
DEFINITION MODULE InOut;

(*      @(#)InOut.def   1.10    91/01/24        SMI     *)

  CONST   EOL = 12C; (* Unix line separator *)

  VAR
    Done   : BOOLEAN; (* Indicates prev operation's Success   *)
    termCH : CHAR;    (* The Char which terminated ReadString *)

  PROCEDURE Read (VAR ch : CHAR );
  (* Done == NOT end-of-file on reading. If true, ch is a     *)
  (* valid character, or is EOL.  If false, ch is 0C and end  *)
  (* of file has been reached.                                *)

  PROCEDURE ReadString (VAR s : ARRAY OF CHAR);
  (* Ignoring leading white space, read a sequence of chars   *)
  (* not including blanks or control chars.  The char which   *)
  (* follows the string is assigned to termCH.  It will not   *)
  (* be seen in subsequent call to any Read routine.          *)

  PROCEDURE ReadInt (VAR x : INTEGER);
  (* Combination of ReadString and conversion to INTEGER.     *)
  (* Syntax : ["+"|"-"] digit { digit } .  Leading blanks are *)
  (* ignored.  Variable termCH is set as in ReadString.       *)
  (* Done == "an integer was successfully read"               *)

  PROCEDURE ReadCard (VAR x : CARDINAL);
  (* Combination of ReadString and conversion to CARDINAL.    *)
  (* Syntax : digit { digit } .  Leading blanks are ignored.  *)
  (* Variable termCH is set as in ReadString.                 *)
  (* Done == "a cardinal was successfully read"               *)

(*      Output Procedures                                     *)

  PROCEDURE Write (ch : CHAR);
  (* Put ch to the output stream.  Write (EOL) is equivalent  *)
  (* to WriteLn.  Other control characters have terminal-     *)
  (* dependent effects.                                       *)

  PROCEDURE WriteLn;
  (* Write a line separator character (EOL).  Under normal    *)
  (* conditions, this will send both carriage return and      *)
  (* line feed to tty device, or the EOL character to a file. *)

  PROCEDURE WriteString (s : ARRAY OF CHAR);
  (* Write the string s to the output stream.  Embedded EOL   *)
  (* characters have the same effect as in Write.  Other      *)
  (* control characters have terminal-dependent effects.      *)

  PROCEDURE WriteInt (x : INTEGER; n : CARDINAL);
  (* Write the integer x with at least n characters of width. *)
  (* Leading zeroes are suppressed.  If n is greater than the *)
  (* needed number of columns, the number is right-justified. *)

  PROCEDURE WriteCard (x : CARDINAL; n : CARDINAL);
```

```
  (* Write the cardinal x with at least n characters of width.*)
  (* Leading zeroes are suppressed.  If n is greater than the *)
  (* needed number of columns, the number is right-justified. *)

END InOut.
```

DEFINITION MODULE MathLib;

```
(*      @(#)MathLib.def 1.8     91/01/24        SMI     *)

PROCEDURE Sqrt    (real : REAL) : REAL;
PROCEDURE Exp     (real : REAL) : REAL;
PROCEDURE Ln      (real : REAL) : REAL;
PROCEDURE Sin     (real : REAL) : REAL;    (* radians *)
PROCEDURE Cos     (real : REAL) : REAL;
PROCEDURE Arctan  (real : REAL) : REAL;
PROCEDURE Entier  (real : REAL) : INTEGER; (* trunc toward neg inf *)
PROCEDURE Power   (real : REAL;
                   exp  : REAL) : REAL;    (* real to the power exp *)
END MathLib.
```

DEFINITION MODULE Storage;

```
(** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
* Module        : Storage
* Project       : Modula-2 standard library
*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*
* Author        : Jirka Hoppe
* Creation date : 1 Jun 1985
* Version       : 1.8
* Copyright     : Oce - Wissenschaftliches Forschungsinstitut AG
*                 Zurich Switzerland, 1985
* Description   : memory allocation
*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Modified by   : Jirka Hoppe
* Mod. date     : 22. August 1985
* Reference     :
* Object        :
* Description   : Procedure 'Change' removed
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

  FROM SYSTEM IMPORT
    ADDRESS;

  EXPORT QUALIFIED
    Allocate, ALLOCATE, DEALLOCATE;

  PROCEDURE Allocate (VAR adr: ADDRESS; units: CARDINAL);

  PROCEDURE ALLOCATE(VAR adr: ADDRESS; units: CARDINAL);

  PROCEDURE DEALLOCATE (VAR adr: ADDRESS; units: CARDINAL);

END Storage.
```


DEFINITION MODULE String;

```
(*      @(#)String.def  1.8     91/01/24        SMI     *)

TYPE
    CompareResult = (less, equal, greater);

PROCEDURE Length    (    str       : ARRAY OF CHAR)  (* in this string      *)
                                   : CARDINAL;       (* count of data chars *)


PROCEDURE Assign    (    source    : ARRAY OF CHAR;  (* copy from this str  *)
                     VAR dest      : ARRAY OF CHAR;  (* to this string      *)
                     VAR success   : BOOLEAN);


PROCEDURE Insert    (    source    : ARRAY OF CHAR;  (* insert this string  *)
                     VAR dest      : ARRAY OF CHAR;  (* into this string    *)
                         index     : CARDINAL;       (* before this char pos*)
                     VAR success   : BOOLEAN);


PROCEDURE Delete    (VAR str       : ARRAY OF CHAR;  (* from this string    *)
                         index     : CARDINAL;       (* 1st char to delete  *)
                         len       : CARDINAL;       (* cnt of chars to del *)
                     VAR success   : BOOLEAN);


PROCEDURE Position  (    pattern   : ARRAY OF CHAR;  (* search for this one *)
                         source    : ARRAY OF CHAR;  (* within this one     *)
                     VAR index     : CARDINAL;       (* 1st char of match   *)
                     VAR found     : BOOLEAN);


PROCEDURE Substring (    source    : ARRAY OF CHAR;  (* copy from this str  *)
                         index     : CARDINAL;       (* at this char postn  *)
                         len       : CARDINAL;       (* for this char count *)
                     VAR dest      : ARRAY OF CHAR;  (* into this string    *)
                     VAR success   : BOOLEAN);


PROCEDURE Concat    (    source1   : ARRAY OF CHAR;  (* this string with    *)
                         source2   : ARRAY OF CHAR;  (* this str on the end *)
                     VAR dest      : ARRAY OF CHAR;  (* assigned to this str*)
                     VAR success   : BOOLEAN);


PROCEDURE Compare   (    string1   : ARRAY OF CHAR;  (* compare this string *)
                         string2   : ARRAY OF CHAR)  (* to this string      *)
                                   : CompareResult;

END String.
```