1. Betrakta följande programavsnitt:

```
PROCEDURE dubbla(n:INTEGER);
BEGIN
  n:=2*n
END dubbla;

VAR x,i:INTEGER;

BEGIN
  x:=1;
  FOR i:=1 TO 5 DO dubbla(x); WriteInt(x,3) END;
  ...
END
```

Meningen var att programmet skulle skriva ut talen `2 4 8 16 32`, men det har visst blivit något fel på proceduren `dubbla`.

a. Vad skriver programmet ut? *(3 poäng)*

b. Rätta till definitionen av proceduren `dubbla` så att programmet fungerar som det var tänkt. *(3 poäng)*

2.

a. Skriv en procedur `ReadLine`, som fungerar som proceduren `ReadString` i modulen `InOut` förutom att den läser in en hel rad istället för bara ett ord. Rader avslutas av tecknet `12C`. Proceduren ska läsa fram till radslut även om inte hela raden kan lagras. *(6 poäng)*

b. Gör en implementering av proceduren `ReadCard` som finns i modulen `InOut`.

(6 poäng)

Notera: procedurerna behöver inte sätta `Done` rätt.

3.

a. Definiera lämpliga typer för att representera datum (år, månad, dag). Använd en uppräkningstyp för månader och en delintervallstyp för dagar. *(3 poäng)*

b. Gör en procedur som skriver ut datum på formatet "*dag månadsnamn årtal*". Utskrift av datumet för denna tenta ska ge

```
29 Maj 1995
```

(5 poäng)

4.

a. Skriv en procedur för att vända på strängar. Låt proceduren ha följande huvud:

```
PROCEDURE RevString(VAR s:ARRAY OF CHAR);
```

Om man t ex vänder på strängen "sirap" ska man få "paris". *(6 poäng)*

b. Skriv en funktion som beräknar fakulteten $n! = n*(n-1)*...*2*1$ av ett heltal n.

(5 poäng)

5. Skriv ett program som läser in en text från terminalen och räknar hur många gånger varje bokstav förekommer i texten. Stora och små bokstäver räknas som lika. Andra tecken struntar man i. Texten avslutas med filslut. Resultatet ska presenteras med ett histogram där varje bokstav som förekommer minst en gång visas som en stapel av *:or (en *:a för varje förekomst av bokstaven).

Exempel: om man matar in texten `Abra Kadabra` ska programmet skriva ut

```
A *****
B **
D *
K *
R **
```

(10 poäng)

6. Denna uppgift går ut på att göra ett program som läser in en text och räknar hur vanliga olika ord är. Ett naturligt sätt att göra detta är att använda en datatyp för *påsar* (eng. bag, multiset). En påse är som en mängd, förutom att antalet förekomster av ett värde har betydelse i påsar.

a. Gör implementeringsdelen till följande modul för påsar:

```
DEFINITION MODULE StringBag;

TYPE StringBag; (* En abstrakt typ för påsar av stängar *)

TYPE String = ARRAY [1..20] OF CHAR;

PROCEDURE NewStringBag(VAR b:StringBag);
(* Skapar en tom påse *)

PROCEDURE InsertString(s:String; VAR b:StringBag);
(* Lägger till strängen s i påsen b *)

PROCEDURE CountString(s:String; b:StringBag):CARDINAL;
(* Ger antalet förekomster av strängen s i påsen b *)

PROCEDURE WriteStringBag(b:StringBag);
(* Skriver ut en påses innehåll.
  För varje sträng i påsen skrivs strängen och antalet förekomster ut. *)

END StringBag.
```

Använd binära sökträd för att lagra påsar. Låt varje nod innehålla en sträng och antalet förekomster av strängen. Första gången en viss sträng läggs i en påse skapas en ny nod i trädet och antal förekomster sätts till 1. Om samma sträng läggs till fler gånger ökas bara antalet förekomster. *(10 poäng)*

b. Gör ett huvudprogram som använder påsmodulen för att räkna ord i en text. Texten avslutas med filslut.

Exempel: om man matar in texten `Fisk och och och och kött` ska programmet skriva ut:

```
Fisk 1
kött 1
och 5
```

(3 poäng)

Biblioteksmoduler

DEFINITION MODULE InOut;

```
PROCEDURE Read (VAR x : CHAR);
(* Read the next character from std input into 'x' *)

PROCEDURE ReadString (VAR x : ARRAY OF CHAR);
(* Read the next string from std input into 'x'. *)
(* Leading blanks are ignored.                    *)
(* Input is terminated by any character <= ' '    *)

PROCEDURE ReadCard (VAR x : CARDINAL);
(* Read the next string from std input and *)
(* convert it to cardinal 'x'.             *)
(* Syntax : digit {digit}                  *)

PROCEDURE ReadInt (VAR x : INTEGER);
(* Read the next string from std input and *)
(* convert it to integer 'x'.              *)
(* Syntax : ['+'|'-'] digit {digit}        *)

PROCEDURE ReadReal (VAR x : REAL);
(* Read the next string from std input and convert it   *)
(* to real 'x'.                                          *)
(* Syntax : ['+'|'-'] digit {digit} ['.' digit {digit}] *)
(*          ['E'['+'|'-'] digit {digit}]                *)

PROCEDURE Write (x : CHAR);
(* Write character 'x' onto std output *)

PROCEDURE WriteString (x : ARRAY OF CHAR);
(* Write the string 'x' onto std output *)

PROCEDURE WriteCard (x : CARDINAL; n : CARDINAL);
(* Convert the cardinal 'x' into decimal representation and *)
(* write it onto std output. Field width is at least 'n'.   *)

PROCEDURE WriteInt (x : INTEGER; n : CARDINAL);
(* Convert the integer 'x' into decimal representation and *)
(* write it onto std output. Field width is at least 'n'.  *)

PROCEDURE WriteReal (x : REAL; n : CARDINAL; k : INTEGER);
(* Convert the real 'x' into external representation and  *)
(* write it onto std output. Field width is at least 'n'. *)
(* If k > 0 use k decimal places.                          *)
(* If k = 0 write x as integer.                            *)
(* If k < 0 use scientific notation.                       *)

PROCEDURE WriteLn;
(* Write the end of line character onto std output *)
(* Emit buffer contents immediately                *)

PROCEDURE WriteBf;
(* Emit buffer contents immediately *)

PROCEDURE Done () : BOOLEAN;
(* last operation ok *)
```

```
   PROCEDURE EOF () : BOOLEAN;
   (* EOF at standard input *)

END InOut.
```

DEFINITION MODULE MathLib;

```
   PROCEDURE sqrt (x : REAL) : REAL;
      (* calculates the square root of 'x' *)

   PROCEDURE sqrtL (x : LONGREAL) : LONGREAL;
      (* calculates the square root of 'x' *)

   PROCEDURE exp (x : REAL) : REAL;
      (* calculates 'e' to the power of 'x', 'e' Euler's number *)

   PROCEDURE expL (x : LONGREAL) : LONGREAL;
      (* calculates 'e' to the power of 'x', 'e' Euler's number *)

   PROCEDURE ln (x : REAL) : REAL;
      (* calculates natural logarithm of 'x' *)

   PROCEDURE lnL (x : LONGREAL) : LONGREAL;
      (* calculates natural logarithm of 'x' *)

   PROCEDURE sin (x : REAL) : REAL;
      (* calculates sine of 'x' *)

   PROCEDURE sinL (x : LONGREAL) : LONGREAL;
      (* calculates sine of 'x' *)

   PROCEDURE cos (x : REAL) : REAL;
      (* calculates cosine of 'x' *)

   PROCEDURE cosL (x : LONGREAL) : LONGREAL;
      (* calculates cosine of 'x' *)

   PROCEDURE arctan (x : REAL) : REAL;
      (* calculates arc tangent of 'x' *)

   PROCEDURE arctanL (x : LONGREAL) : LONGREAL;
      (* calculates arc tangent of 'x' *)

   PROCEDURE real (x : INTEGER) : REAL;
      (* converts 'x' to type 'REAL' *)

   PROCEDURE realL (x : INTEGER) : LONGREAL;
      (* converts 'x' to type 'LONGREAL' *)

   PROCEDURE entier (x : REAL) : INTEGER;
      (* calculates the largest integer <= 'x' *)

   PROCEDURE entierL (x : LONGREAL) : INTEGER;
      (* calculates the largest long integer <= 'x' *)

END MathLib.
```

```
DEFINITION MODULE Storage;
(****************************************************************************)
(* Copyright (c) 1988 by GMD Karlruhe, Germany      *)
(* Gesellschaft fuer Mathematik und Datenverarbeitung      *)
(* (German National Research Center for Computer Science)      *)
(* Forschungsstelle fuer Programmstrukturen an Universitaet Karlsruhe      *)
(* All rights reserved.         *)
(* Don't modify this file under any circumstances      *)
(****************************************************************************)

   FROM SYSTEM IMPORT ADDRESS;

   PROCEDURE ALLOCATE (VAR a : ADDRESS; size : CARDINAL);
      (* Allocates an area of the given size 'size' and returns it's *)
      (* address in 'a'. If no space is available, 'a' becomes 'NIL'. *)

   PROCEDURE DEALLOCATE (VAR a : ADDRESS; size : CARDINAL);
      (* Frees the area of size 'size' starting at address 'a'. *)
      (* Upon return 'a' is set 'NIL' *)

END Storage.
```

```
DEFINITION MODULE String;
(***************************************************************************)
(* Copyright (c) 1988 by GMD Karlruhe, Germany      *)
(* Gesellschaft fuer Mathematik und Datenverarbeitung      *)
(* (German National Research Center for Computer Science)      *)
(* Forschungsstelle fuer Programmstrukturen an Universitaet Karlsruhe      *)
(* All rights reserved.        *)
(* Don't modify this file under any circumstances      *)
(***************************************************************************)

   TYPE String = ARRAY [0..255] OF CHAR;

   PROCEDURE EmptyString (VAR str: ARRAY OF CHAR);
      (* str := "" *)

   PROCEDURE Assign (VAR dst, src: ARRAY OF CHAR);
      (* assign string 'src' to string 'dst'. 'src' must be terminated by 0C *)

   PROCEDURE Append (VAR dest, suffix: ARRAY OF CHAR);
      (* append 'suffix' to 'dest', only significant characters. *)

   PROCEDURE StrEq (VAR x, y: ARRAY OF CHAR): BOOLEAN;
      (* x = y , only significant characters. *)

   PROCEDURE Length (VAR str : ARRAY OF CHAR) : CARDINAL;
      (* returns the number of significant characters. *)

   PROCEDURE Insert
      (substr: ARRAY OF CHAR; VAR str: ARRAY OF CHAR; inx: CARDINAL);
      (* Inserts 'substr' into 'str', starting at str[inx] *)

   PROCEDURE Delete (VAR str: ARRAY OF CHAR; inx, len: CARDINAL);
      (* Deletes 'len' characters from 'str', starting at str[inx] *)

   PROCEDURE pos (substr: ARRAY OF CHAR; str: ARRAY OF CHAR): CARDINAL;
      (* Returns the index of the first occurrence of 'substr' in 'str' or *)
      (* HIGH (str) + 1 if 'substr' not found. *)

   PROCEDURE Copy
      (str: ARRAY OF CHAR; inx, len: CARDINAL; VAR result: ARRAY OF CHAR);
      (* Copies 'len' characters from 'str' into 'result', *)
      (* starting at str[inx] *)

   PROCEDURE Concat
      (s1, s2: ARRAY OF CHAR; VAR result: ARRAY OF CHAR);
      (* Returns in 'result' the concatenation of 's1' and 's2' *)

   PROCEDURE compare (s1, s2: ARRAY OF CHAR): INTEGER;
      (* Compares 's1' with 's2' and returns -1 if s1 < s2, 0 if s1 = s2, *)
      (* or 1 if s1 > s2 *)

   PROCEDURE CAPS (VAR str: ARRAY OF CHAR);
      (* CAP for the entire 'str' *)

END Strings.
```