1. I kursen har vi tittat på olika sätt att lagra och söka i information. En laborationsuppgift gick ut på att jämföra två olika implementeringar av en telefonkatalog.

Olika datastrukturer och metoder ger olika snabba program. Hur snabbt går insättning och sökning i genomsnitt med normal implementering av följande metoder?

a. En osorterad ARRAY. Nya element läggs till sist.

b. En osorterad länkad lista.

c. En sorterad ARRAY. Nya element sätts in på rätt plats. Vid sökning utnyttjar man att elementen är sorterade.

d. Binära sökträd.

Svara genom att placera in metoderna på rätt plats i följande tabell: *(8 poäng)*

	Sökning O(n)	Sökning O(log n)	Sökning O(1)
Insättning O(n)			
Insättning O(log n)			
Insättning O(1)			

Förklaring av beteckningarna:

- O(n) = tiden blir proportionell mot antalet element i datastrukturen.
- O(log n) = tiden blir proportionell mot logaritmen av antalet element i datastrukturen.
- O(1) = operationen går alltid lika fort, oberoende av hur många element det finns i datastrukturen.

2. Skriv en procedur som testar om en sträng är ett palindrom. Låt proceduren ha följande huvud:

```
PROCEDURE IsPalindrome(s:ARRAY OF CHAR):BOOLEAN;
```

En sträng är ett palindrom om den är likadan framlänges som baklänges, bortsett från blanktecken och skiljetecken. Stora och små bokstäver räknas som lika. Exempel:

```
Was it a car or a cat I saw?
Saippuakauppias
Aibohphobia
```

Saippuakauppias *(tvålförsäljare på finska)*

Aibohphobia *(palindromskräck)*

(8 poäng)

3.

a. Definiera lämpliga typer för att representera information om atleter. Följande information är av intresse: namn, land, födelsedatum, födelseort, längd, vikt, skomärke, gift/ogift. Antag att det finns en färdig typ `Date` för datum. Land lagras som en trebokstavskod. *(4 poäng)*

b. Gör en procedur som skriver ut information om en atlet formaterad som i följande exempel:

```
Country: SWE
Name   : Patrik SJOBERG
Birth  : 05 January 1965 Goteborg(SWE)
Height :  2.00 m
Weight :  82.0 kg
Shoes  : ASICS TIGER
Not married
```

Längd ges alltså i meter och skrivs ut med två decimaler. Vikten skrivs ut med en decimal. I exemplet är "`Goteborg(SWE)`" födelseorten.

Antag att det finns en färdig procedur för att skriva ut datum:

```
PROCEDURE WriteDate(d:Date);
```

(6 poäng)

4. För att lättare kunna hålla reda på innehållet på sina videoband håller Movitz Fanatikus på att göra ett program. Han har redan gjort en modul med en abstrakt datatyp för filmer:

```
DEFINITION MODULE Movie;

TYPE Movie; (* En abstrakt typ för information om filmer *)

TYPE Minutes = CARDINAL; (* En typ för tider. Tider mäts i minuter. *)

PROCEDURE ReadMovie(VAR m:Movie); (* läser in information om en film *)
PROCEDURE WriteMovie(m:Movie);    (* skriver ut en film *)

PROCEDURE MovieLength(m:Movie):Minutes; (* ger en films längd *)

(* ... *)

END Movie.
```

Movitz vill nu ha hjälp med vissa delar av programmet. Han behöver en modul med typer för videoband. Modulen behöver tillhandahålla:

- En typ för videoband. För varje band behöver man veta dess längd (i minuter) och vilka inspelningar som är gjorda på bandet.
- En typ för listor av inspelningar. För varje inspelning behöver man veta vilken film som är inspelad och var på bandet (räknat i minuter från bandets början) inspelningen börjar. Denna typ ska vara abstrakt.
- En procedur för att hitta ett ledigt ställe på ett band när man ska spela in en ny film. Proceduren har ett band och en film som argument. Resultatet ska vara att filmen läggs till listan av inspelningar på bandet samt att man får reda på (via en extra `VAR`-parameter) var på bandet man ska spela in filmen. Det behövs också ett sätt att få reda på om filmen inte får plats på bandet.

  Notera att Movitz ibland tröttnar på filmer, så det kan finnas luckor mellan inspelningarna på ett band. Det gäller alltså att leta upp en lucka där den nya filmen får plats.

a. Skriv definitionsdelen till videobandsmodulen. *(6 poäng)*

b. Skriv implementeringsdelen till videobandsmodulen. Använd en länkad lista för listor av inspelningar. Det är lämpligt låta inspelningarna ligga i samma ordning i listan som på bandet. *(12 poäng)*

5. Gör en enkel miniräknare, d v s ett program som läser in enkla aritmetiska uttryck och beräknar deras värde. Exempel på användning (text i *kursiv* stil är det som användaren matar in):

```
4 + 2 * 3 =
18
2 * 3 + 4 =
10
```

De operationer som kan utföras är de fyra räknesätten. Operander och resultat är heltal. Alla operatorer har samma prioritet och uttrycken beräknas från vänster till höger.

Man kan även använda variabeln `it` som operand. Denna står då för värdet av det senast beräknade uttrycket:

```
5 - 2 =
3
10 * it * it =
90
```

Nedan följer en EBNF-grammatik för indata till programmet:

```
Indata   = { Uttryck }.
Uttryck  = Operand { Operator Operand } "=".
Operand  = "it" | Number.
Operator = "+" | "-" | "*" | "/".
```

Det ska alltid finnas blanktecken mellan operatorer och operander (så att man kan läsa in dem med `ReadString`).

Om programmet får indata som inte följer den ovan givna syntaxen ska det skriva ut ett felmeddelande och stoppa. (Man kan stoppa programmet genom att anropa den inbyggda proceduren `HALT`.)

Det finns en modul med hjälpfunktioner som du kan använda.

```
DEFINITION MODULE Numbers;

PROCEDURE IsNumber(s:ARRAY OF CHAR):BOOLEAN;

PROCEDURE StringToNum(s:ARRAY OF CHAR):CARDINAL;

END Numbers.
```

`IsNumber` undersöker om alla tecken i en sträng är siffror. `StringToNum` konverterar ett (icke negativt) heltal representerat som en sträng till typen `CARDINAL`. Det är meningen att man ska använda `IsNumber` för att kontrollera en sträng innan man anropar `StringToNum`, så `StringToNum` gör ingen felkontroll.

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

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

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

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

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

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

   PROCEDURE entier (x : REAL) : INTEGER;
      (* calculates the largest 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.
```