CODEN: LUTFD2/(TFRT-5394)/1-73/(1988)

Expert Systems for Planning of Hydro Power Production

Olof Wickström

Department of Automatic Control
Lund Institute of Technology
December 1988

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Master Thesis

Date of issue
December 1988

Document Number
CODEN: LUTFD2/(TFRT-5394)/1-52/(1989)

Author(s)
Olof Wickström

Supervisor
Jan Eric Larsson

Sponsoring organisation

Title and subtitle
Expert Systems for Planning of Hydro Power Production.

Abstract

This report describes different ways of using expert system techniques when making programs that should simplify the planning of water flows through harnessed rivers. An expert program and a critic program has been implemented in Scheme and Prolog respectively. Two other ways to implement expert systems (Pascal and KEE) have also been studied.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
73

Recipient's notes

Security classification

Contents:

Appendices

0. Introduction

This Master Thesis has been written during the summer and autumn of 1988 at Sydkraft AB and at the Department of Automatic Control, Lund Institute of Technology.

The project, which can be divided in three quite distinct parts: the identification and classification of knowledge, the examination of different ways of implementation, and the implementation of a primitive expert and of an equally primitive critic, is intended to study different ways of using expert system techniques when planning the water flow through the power stations and reservoirs in Sydkraft's power generation system

It must already here be stressed that the written expert program is not a traditional expert system i.e. consists of a database with if-then-else rules and an inference engine using these rules to see if something is true. Instead all the rules are functions which are applied to the data on the data's way from input to output. The critic is more like traditional expert systems program but its rules works on numerical data.

In the same way as the project was divided this paper can be divided in three parts: in chapter one through three the problem is described and the available knowledge is classified, in chapter four different ways to implement expert systems are studied, and in chapter five and six respectively, an expert program and a critic program is described with a discussion of possible further developments in chapter seven. The conclusions are to be found in chapter eight.

I would here like to express my thanks to my two supervisors, Bengt-Tore Sondh of Sydkraft AB and Jan Eric Larsson of the Department of Automatic Control, for their invaluable help and support.

1. Two Important Concepts

Vattendomar

The swedish laws that govern the harnessing of rivers states that before any work can take place the plans must be put forward the water rights court, in Swedish Vattendomstolen. This court consist of jurists, representatives from the local population and representatives from the company which wants to harness the river. These makes an inquiry of the effect of the harnessing on fishing, tourism, farming, etc. Based on this inquiry they then writes the water rights, in Swedish Vattendomar. Henceforth, whenever I will use the word water rights I will mean this Swedish version.

These water rights do not only regulate maximal and minimal flow through the stations and the maximal and minimal water level in the reservoirs but also stipulate what the power company has to do to minimize the damage the harnessing would do to all kinds of interests.

The problem with the water rights is that the water rights court's composition often leads to peculiar water rights. For instance is it rather usual that the water rights stipulates certain water flows on major holidays. To further complicate things the water rights could be as old as from the first decades of this century. This gives trouble as the society around the rivers has changed quite a bit since they were written[1].

The Gränskraftpriskurva

This is the major central control instrument when making plans for the usage of different power sources at Sydkraft. It is the expected energy price for the last in the system generated megawatt each hour in the week and it is a function of the expected power requirements. Its importance lies in the fact that it is the only short time plan that is made for the whole hydro power generating system. Henceforth I will call it pricecurve[2].

2. Water Planning

The main purpose of the planning of the water flows through Sydkraft's power stations and reservoirs, henceforth water planning, is to get as much power as possible from the available water.

The Timescale

The water planning is done in three timescales: yearly, weekly, and daily. This study has its focus on the weekly planning but parts of it can be applied on a more general program. The reason for this is that the shorter timespan the plans are made for the more written rules exist, the making of yearly plans involves a lot of experience based prognostication. The choice of weekly planning instead of daily is based on that the daily planned stations are run using a real time control system which is unable to communicate with other computer systems. Therefore have I in this study used Järnvägsforsens power station in the river Ljungan[3], which is not run in real time, as example when writing my programs.

Problems

One of the problems when making water plans is to stay inside the limits that are put up by the water rights at the same time as the water plan should be so efficient as possible. The knowledge needed for this is further described in the next chapter but it can already be unveiled that it consists of a lot of rules.

How It Is Done Today

Today when doing the weekly planning the planner has to his aid the rules which are described in the next chapter, the average water flow for the week, which he gets from the yearly plans, and the pricecurve. With these aids the planner uses a spreadsheet to make the water plans. The expert's main problem when doing this is to remember all the rules. This is further stressed by the fact that shift working and yearly variations in the water flows makes the time between the times the expert has to consider the same rules several years.

3. The Different Kinds of Knowledge Needed in Water Planning

The knowledge at Sydkraft about how to plan the water flow through their power stations and reservoirs, henceforth water planning, can be divided in three distinct sets of rules.

First there are the so called water rights. These are written by the water rights court before any waterfall could be harnessed and are legal documents. As such they are easily assecible. Henceforth this class of rules will be called class A.

The second set of rules consists of rules that are more or less empirically gained. The set consists of rules that gives advice about how to plan the water flow so that the the water rights should not be violated and other useful things, such as by a certain flow through a certain station the turbine starts to oscillate with its natural frequency. Also these rules are easily assecible as the information are kept in folders at Sydkraft. Henceforth this class of rules will be called class B.

The sets of rules above has in common that each of the rules are valid for only one or a couple of stations and reservoirs.

The rules of the third set are quite different from the rules above, they are the rules which describe a "good" system. These rules are personal for each operator even if there naturally exists an unofficial department standard. The rules are very complex and dependant of a lot of variables. In general the operators can easily decide if a solution is bad but they can not always tell why. Henceforth this class of rules will be called class C.

Examples of these rules can be found in sections 5 and 6.

4. Ways of Implementation

Four tools for implementation of an expert system has been studied, three languages and an expert system development tool. The languages are Pascal, Prolog and Scheme (a Lisp dialect) and the development tool is KEE. The reason for choosing these is mainly that they have been available to me.

Pascal

The advantages of using Pascal as an expert system language are:

It is widely spread, there exists a Pascal compiler for every *real* computer, except Lisp machines, which gives programs written in it a large portability.

The expert, who probably has some knowledge of Algol oriented languages, can easily understand the code, which makes it possible for him to suggest, or even self make, improvements in the code.

It is rather efficient.

It is rather easy to construct complicated datastructures, especially if these do not have to contain any rules.

The disadvantages are:

It is not made for construction of expert systems and therefore contains nothing to support any such activities. This results in very complicated if-then-elseif... statements whenever one tries to implement rules.

Another big disadvantage is that the code are so strongly separated from data.

Prolog

The logic programming language Prolog is naturally very good for solving problems that can be expressed by using predicate calculus. This makes it easy to implement rules of the form if A and B and... then X. The language's inherent mechanism for backtracking takes care of parts of the programmer's problem to write program driving rules. The language's rather unique programing style makes it necessary for the expert to study it rather thoroughly before he can understand the program code. Another disadvantage is Prolog's limited set of data structures, there only exists one: lists, and the fact that the only operations you can do on these lists are Head and Tail (These divides a list in its first element and the rest). Other disadvantages are the low portability which comes from that there exists no language standard and that it is not so efficient for numerical calculations[4].

Lisp

The list processing language LISP has much more datastructures than Prolog and a lot of useful operations on them. It even has some possibilities for object orientation through its property lists. Many Lisp dialects has non standardized overlays, some times called flavors, which supports object orientation (For instance: when editing this thesis I have got information of the CLOS's (Common Lisp Object System's), an object orientated overlay, acceptance as and inclusion in Common Lisp). At the same time the language does not make any difference between functions and variables, a very useful feature. Among its drawbacks is its large demand for computational power on ordinary computers[5].

KEE

The Knowledge Engineering Environment is an expert system development tool from IntelliCorp. Its graphics are very good and it is very usable for making structured Lisp programing but its knowledge representation is quite complex, and I did not have the time to get a good grip of it during the project[6].

The strong overweight of languages depends, as stated previously, on that they were available to me. What was not available but I would have liked to study are an object orientated language like Smalltalk and a smaller tool like Nexpert Object.

5. The Planner

This expert program tries to imitate the way a real expert plans the water flow through a station. He starts by giving the most costly hours, accordingly to the pricecurve, the highest practical flow, in the case of Järnvägsforsen the flow with the highest efficiency when using both turbines: 120 m^3/s. This is done to get the highest possible power out of the water. The amount of water that are placed in this way depends on what the expert thinks he will do in his next step. In this he adjusts the flows he already has planned and adds some new one to fulfil the class A and class B rules. Doing this he naturally also looks at the class C rules.

The program tries to imitate this behavior but it only uses these rules:

1) The reservoir available can only hold 600 t.e. (1 t.e. is the volume that by a flow of 1 m^3/s would pass during 1 hour i.e. 3.600 m^3.) This is a typical class A rule.

2) If the sum of the flow through Järnvägsforsen and Torpshammar is more than 200 m^3/s for more than ten hours then the flow through Järnvägsforsen can not be zero for more than seven hours afterwards. This is to make sure that one of the water rights concerning Torpsjön, this makes this rule a class B rule.

3) If you are using both turbines you would not stop for just one hour and then start them again, instead you would run over this single hour without noticing it. This rule and the next is of class C.

4) You will have to wait at least half an hour after you have started/stopped one unit before you start/stop the other

The idea to make the program imitate a persons behavior makes traditional expert systems programming techniques unsuitable. Instead I have used a more traditional programming style where the rules are represented in functions which does a good deal of work when applied to data.

Flow of Control

To get an idea about the program's flow of control let us take a look at the main program:

```
(define (mainprogram)
  (do
    ((str "j" str))
    ((not (member str (list "j" "J" "y" "Y" "ja" "Ja" "JA"
    "yes" "Yes" "YES"))))
```

```
(init)
```
Initiate stuff

```
(place_minflows station)
```
This is to make it possible to define the minimal flow a certain hour. It is to be used when implementing rules that put constraints on this matter.

```
(let work ((price (car pricelist)) (list_a (cdrpricelist))
   (x (- 1 (read_x))))
   (cond
      ((null? list_a) ())
      ((> Amount_of_water (* x Initial_amount_of_water))
         (place_evenly Week price x)
         (work (car list_a) (cdr list_a) x))))
```
This loop places the maximal flow on the highest priced hours until x of the available water is placed.

```
(clean_up Station)
```
Takes care of the third rule.

```
(apply_reservoir_rules Week)
```
Reduces the flows that was planned in (place_evenly) and gives the most efficient flow to some henceforth unplanned hours to fulfil the first rule.

```
(print_flows Week)
```
Writes the result.

```
(display "One more time?")
(set! str (symbol->string (read)))
(newline))
)
```

To further explain how the program works I will, on the next two pages, give some snapshots from an execution.

```
>> (mainprogram)
Which station?jfn
What is the average flow?33
What is the initial amount of water?0
Which is the first day of the week (yymmdd)?880810
In which file is the pricelist kept? (prislista.dat)"prislista.dat"
How big part of the water should be placed?0.9
```

day 1			day 2			day 3			day 4			day 5			day 6			day 7		
con	flw	price	con	flw	price	con	flw	price	con	flw	price	con	flw	price	con	flw	price	con	flw	price
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	0	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0	50	0	0	50
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	0	90	0	0	90	0	0	90	0	0	90	0	0	90	0	0	90	0	0	90
0	0	90	0	0	90	0	0	90	0	0	90	0	0	90	0	0	90	0	0	90
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	30		0	30		0	30		0	30		0	30		0	30		0	30	

con = reservoir contents in t.e.
flw = flow
price = price

Here we have started with an average flow of 33 m^3/s and an empty reservoir. After the loop which contains place_evenly the program has placed 90% of the available water on the most costly hours.

day 1			day 2			day 3			day 4			day 5			day 6			day 7		
con	flw	price	con	flw	price	con	flw	price	con	flw	price	con	flw	price	con	flw	price	con	flw	price
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	32	0	0	32	0	0	32	0	0	32	0	0	32	0	0	32	0	0	32	0
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	120	50	0	120	50	0	120	50	0	120	50	0	120	50	0	120	50	0	120	50
0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90	0	120	90
0	32	90	0	32	90	0	32	90	0	32	90	0	32	90	0	32	90	0	32	90
0	0	90	0	0	90	0	0	90	0	0	90	0	0	90	0	0	90	0	0	90
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	37		0	37		0	37		0	37		0	37		0	37		0	37	

After clean_up we have this situation. Clean_up has started and stopped one turbine, of the two in Järnvägsforsen, 30 minutes before the other by placing 32 m^3/s flows. It has also skipped the one hour stops.

day 1			day 2			day 3			day 4			day 5			day 6			day 7		
con	flw	price	con	flw	price	con	flw	price	con	flw	price	con	flw	price	con	flw	price	con	flw	price
33	0	0	391	0	0	399	0	0	407	0	0	415	0	0	423	0	0	371	0	0
66	0	0	424	0	0	432	0	0	440	0	0	448	0	0	456	0	0	404	0	0
99	0	0	457	0	0	465	0	0	473	0	0	481	0	0	489	0	0	437	0	0
132	0	0	490	0	0	498	0	0	506	0	0	514	0	0	522	0	0	470	0	0
165	0	0	523	0	0	531	0	0	539	0	0	547	0	0	555	0	0	503	0	0
198	0	0	556	0	0	564	0	0	572	0	0	580	0	0	588	0	0	536	0	0
199	32	0	557	32	0	565	32	0	573	32	0	581	32	0	589	32	0	537	32	0
112	120	90	470	120	90	478	120	90	486	120	90	494	120	90	502	120	90	450	120	90
25	120	90	383	120	90	391	120	90	399	120	90	407	120	90	415	120	90	363	120	90
26	32	90	296	120	90	304	120	90	312	120	90	320	120	90	328	120	90	276	120	90
59	0	90	209	120	90	217	120	90	225	120	90	233	120	90	241	120	90	189	120	90
92	0	90	122	120	90	130	120	90	138	120	90	146	120	90	154	120	90	102	120	90
125	0	50	35	120	50	43	120	50	51	120	50	59	120	50	67	120	50	15	120	50
93	65	90	36	32	90	44	32	90	52	32	90	60	32	90	8	92	90	16	32	90
61	65	90	69	0	90	77	0	90	85	0	90	93	0	90	41	0	90	49	0	90
94	0	90	102	0	90	110	0	90	118	0	90	126	0	90	74	0	90	82	0	90
127	0	0	135	0	0	143	0	0	151	0	0	159	0	0	107	0	0	115	0	0
160	0	0	168	0	0	176	0	0	184	0	0	192	0	0	140	0	0	148	0	0
193	0	0	201	0	0	209	0	0	217	0	0	225	0	0	173	0	0	181	0	0
226	0	0	234	0	0	242	0	0	250	0	0	258	0	0	206	0	0	214	0	0
259	0	0	267	0	0	275	0	0	283	0	0	291	0	0	239	0	0	247	0	0
292	0	0	300	0	0	308	0	0	316	0	0	324	0	0	272	0	0	280	0	0
325	0	0	333	0	0	341	0	0	349	0	0	357	0	0	305	0	0	313	0	0
358	0	0	366	0	0	374	0	0	382	0	0	390	0	0	338	0	0	346	0	0
146	18		296	32		304	32		312	32		320	32		301	35		276	32	

```
One more time?n
```

This is the final result where the 120 m^3/s runs have been shorted to avoid emptying the reservoir and 65 m^3/s flows has been placed to avoid getting to much water in the reservoir and to have as much water as possible in the reservoir before the 120 m^3/s runs.

For more information see the code which is kept in appendix D.

The Distinction Between Knowledge and Inference Machine

One of the fundamentals of expert system programming is the easy interchangability of knowledge. This is done by separating the knowledge from the rest of the program. In this program the knowledge is kept in four functions, in initiate_station, in clean_up, in work_on_blocks _station, and in apply_reservoir_rules.

In initiate_station is kept basic numerical data about the powerstations such as the maximal reservoir contents and the possible flows through the power stations.

In clean_up is kept a list of rules that are general for all stations such as the rule that says that you will have to wait at least half an hour after you have started/stopped one unit before you start/stop the other. These rules are kept as functions which are applied one after the other to the plan.

In work_on_blocks_station is kept a list of rules that regulates the main flows through a certain station, also these are applied one after the other to the plan.

In apply_reservoir_rules is kept the rule that says that you should neither keep more water in the reservoir than its maximal possible content nor empties the reservoir.

Implementation

The program has been implemented in Chez Scheme version 2.0 on a VAX 780 working under VMS version 4.5. It took around 180 hours to produce 1460 lines of code.

The major problem during this work has been the lack of good debugging facilities in the interpreter which has made the development of the program unnecessary cumbersome. The other main problem has been to make sure that rules does not nullify previously applied rules when applied to the data.

6. The Examiner

As opposed to the Planner program, which tries to imitate a real expert, the Examiner, a critic program, is an aid to the real expert which checks that his plan does not violate any rule. If it does it gives notice and gives a short description of the error. This makes the programers task much simpler as he does not have to formalize all the class C rules as many of these are unnecessary for a well functioning program. To keep it simple the Examiner does not give any advice when it finds an error. If it should have this feature it would have had to have full knowledge of the class C rules.

The Examiner, as it is written, checks if a water plan is in accordance with three rules:

1) That the available reservoir would not become empty or overfull. A class A rule.

2) That if the sum of the flow through Järnvägsforsen and Torpshammar is more than 200 m/s for more than ten hours then the flow through Järnvägsforsen would not be zero longer than seven hours afterwards. A class B rule.

3) That the station would not be started more than twice daily. A class C rule.

Flow of Control

In a Pascal like fashion the Examiners flow of control could be described as follows:

```
begin
   init
   for each day in the week do
      for each hour of the day do
         allowed
   examine_weekend
   finish
end
```

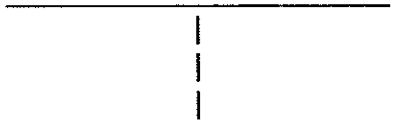
where init initiates stuff, allowed checks if a certain flow is allowed a certain hour, examine_weekend checks the rules that applies to the end of the week and finish closes files and tidies up the database.

Examples of Execution

Here are some output from the program to show how it reacts on different input. The leftmost column is the number of the day thereafter comes the number of the hour and the flow on that hour. These are the program's input along with the planned average flow of the week and the flows through Torpshammar. The fourth column is the content of the reservoir in t.e.

In this first example the input to the program are actual figures from Järnvägsforsen. The program has, not surprisingly, nothing to complain on.

```
Day    Hour   Flow     Contents
100      1      0        288
100      2      0        320
100      3      0        352
100      4      0        384
100      5      0        416
100      6      0        448
100      7      0        480
100      8      0        512
```



```
106     17      0         61
106     18      0         93
106     19      0        125
106     20      0        157
106     21      0        189
106     22      0        221
106     23      0        253
106     24      0        285
Examiner_weekend is not
implemented but can be used
for checking rules that
are valid for weekends.
```

This is the end message which tells that the program has invoked examine_weekend. As the program does not check any rules that applies to the end of the week the program only echoes this message.

In the second example the figures are the same but the day is in the summer when the available reservoir is 600 t.e. accordingly to the water rights. This gives an overfull reservoir and an message in accordance with this.

Day	Hour	Flow	Contents
200	1	0	288
200	2	0	320
200	3	0	352
200	4	0	384
200	5	0	416
200	6	0	448
200	7	0	480
200	8	0	512
200	9	0	544
200	10	0	576
To large contents			
200	11	0	608
To large contents			

206	18	0	93
206	19	0	125
206	20	0	157
206	21	0	189
206	22	0	221
206	23	0	253
206	24	0	285

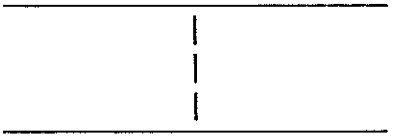
Examiner_weekend is not implemented but can be used for checking rules that are valid for weekends.

In the third example the input has been manipulated in a way so that the turbine is started tree times on day 106. This is against the implemented rules and the program promptly gives an appropriate message.

```
Day    Hour   Flow     Contents
100      1      0        288
100      2      0        320
100      3      0        352
100      4      0        384
100      5      0        416
100      6      0        448
                |
                |
                |
105      23     0        395
105      24     0        427
106      1     65        459
106      2     65        426
106      3     65        393
106      4     0        360
106      5     65        392
106      6     65        359
106      7     65        326
106      8     65        293
106      9     0        260
106      10     65        292
The unit should not be started
more than two times each day.
106      11     65        259
106      12     65        226
106      13     65        193
106      14     65        160
106      15     0        127
106      16     0        159
106      17     0        191
106      18     0        223
106      19     0        255
106      20     0        287
106      21     0        319
106      22     0        351
106      23     0        383
106      24     0        415
Examiner_weekend is not
implemented but can be used
for checking rules that
are valid for weekends.
```

In the last example the sum of the flow through Torpshammar and Järnvägsforsen is more than 220 m^3/s for more than seven hours and Järnvägsforsen stands still more than seven hours thereafter. This violates the third rule in the program and results in an appropriate message. Notice that the program does not check if the reservoir contents is negative and therefore does not give any error message when this happens.

```
Day    Hour  Flow     Contents
100      1     100        288
100      2     100        220
100      3     100        152
100      4     100        84
100      5     100        16
100      6     100        -52
100      7     100        -120
100      8     100        -188
100      9     100        -256
100      10     100        -324
100      11     100        -392
100      12     100        -460
100      13     0         -528
100      14     0         -496
100      15     0         -464
100      16     0         -432
100      17     0         -400
100      18     0         -368
100      19     0         -336
100      20     0         -304
If the flow through thr + jfn
>= 200 for more than ten hours
then Jfn must not be stoped for
more than 7 hours thereafter.
100      21     0         -272
```



```
106      21     0         -881
106      22     0         -849
106      23     0         -817
106      24     0         -785
Examiner_weekend is not
implemented but can be used
for checking rules that
are valid for weekends.
```

For more information see the code which is kept in appendix E.

The Distinction Between Knowledge and Inference Machine

In this program the check of the rules is done by one function for each rule. These functions are called from allowed and examine_weekend one after the other and they do not interact with each other or the rest of the program.

Implementation

The program has been implemented in C-Prolog version 1.5 on a VAX 8530 working under VMS version 4.7. It took only something like 35 hours to write 283 lines.

The program's simple structure makes the programming rather straightforward. The main problem has been to make the interpreter work the way the manual said it should. It showed itself to be necessary to adjust (increase) the global and local stacks to 1 Mbyte and 1,9 Mbyte respectively and the trail to 500 kbyte of the interpreter. This should work fine according to the manual but on the Department of Automatic Control's VAX 780 it did not work at all. A shift to the School of Electrical Engineering's VAX 8530 where the interpreter worked as it should solved the problem.

7. Further Developments

It is possible to further develop both programs by incorporating more rules. The main difference comes in the amount of time needed to get an usable program. The Planner program, where there is a risk of interaction between rules and with it's demand for formalization of the complex class C rules, needs much more time than the Examiner. In the latter the rules does not interact which makes the development effort more or less linear with the number of rules incorporated. It also uses already formalized rules, thereby getting rid of most of the knowledge engineering needed for the development of the Planner. To get a hint of the difference of time needed for further development, one can look at the fact that it took me 180 hours to write the Planner program but only 35 hours to write the Examiner program. The Examiner can also be developed by making it able to read data directly from the spreadsheet that the real expert uses when he makes the water plans. If instead execution time becomes a problem the possibility to rewrite the program into Pascal should be considered.

Another way to develop this project is to look at other ways of implementing expert systems. Previously in this thesis has Smalltalk, an object orientated language, and Nexpert Object, a smaller expert system shell, been mentioned.

8. Conclusions

The possibility of making programs both for planning of water flows and for criticizing plans already made has been shown. The critic has shown itself to be much simpler to implement, and thereby resulting in a smaller project, than the planner. At the same time the critic is as capable to solve the problem that initiated this study as the expert system. The making of a big planner program seems to me to be a big project of little use. Especially as a human expert probably can do the work faster than the machine if he has an aid that notifies him about rules that he has overseen.

Furthermore four different ways to implement expert systems: Pascal, Lisp, Prolog, and KEE, have been studied. In this study the languages have turned up best and the expert system development tool has shown itself to be too complicated to be mastered in the short time that I have had to my disposal.

9. Notes

1 Abstracts from a typical waterright can be found in appendix A.

2 An example of a gränskraftpriskurva can be found in appendix B.

3 A map of the area is found in appendix C.

4 For more information about Prolog see for instance:
Prolog Programming for artificial intelligence, by Ivan Bratko, Addison-Wesley Publishers Limited 1986
or
BDPROLOG-Un implementation de Prolog en Lisp, by Björn Davidsson, EPFL Hiver 87/88 (not published).

5 For more information about LISP see for instance:
Structure and Implementation of Computer Programs, by Harold Abelson and Gerald Jay Sussman with Julie Sussman, MIT Press 1987.

6 For more information about KEE see for instance:
Expert Systems, Artificial Intelligence in Business, by Paul Harmon and David King, John Wiley & Sons Inc 1985

10. Sources

Besides the sources mentioned in the previous chapter I have used the following:

Programing in Prolog, by W. F. Clocksin and C. S: Mellish, Springer-verlag 1984

Artificial Inteligence, by P. H. Winston Addison-Wesley 1977

Introduction to Artificial Inteligence, by E. Charniak and D. McDermolt, Addison-Wesley 1985

Various document in internal use at Sydkraft.

A Abstracts From a Typical Water Right

LJUNGÅNS REGLERINGAR (A 43/61)

TORPSHAMMARS KRAFTSTATION OCH UTVIDGAD KORTTIDSREGLERING I GIMÅN (A 31/68)

Dom angående idrifttagning m.m. meddelad den 22 augusti 1975.

Transumt

-.-

7. VATTENHUSHÅLLNINGSBESTÄMMELSER

-.-

7.4. Sammanfattning av beslutade vattenhushållningsbestämmelser

7.4.1 Nedre Ljungans korttidsreglering

Vattendomstolen fastställer följande vattenhushållningsbestämmelser att gälla för den vid Järnvägsforsens kraftverk bedrivna dygnsreglering, till vilken tillstånd lämnas i denna dom.

A. Korttidsreglering vid Järnvägsforsens kraftverk

1. Med iakttagande av de för Holmsjöns årsreglering gällande bestämmelserna med visst undantag enligt punkt A 2 nedan samt de vid vart tillfälle för Gimåns korttidsreglering gällande bestämmelserna och med iakttagande av vad i övrigt av nedanstående vattenhushållningsbestämmelser framgår, må sökandena utöva dygnsreglering vid Järnvägsforsens kraftverk och därvid utnyttja årsregleringsmagasinet i Holmsjön i sådan omfattning att skillnaden mellan högsta och lägsta vattenstånd i Holmsjön i vad på korttidsregleringen beror per dygn icke överstiger 10 cm.

2. Korttidsreglerad tappning skall framsläppas genom kraftsta-

tionen, varvid vattenföringen får variera mellan 0 och 145 m³/s. För Holmsjöns årsreglering fastställd minimitappning får då dygnsreglering bedrives tappas som dygnsmedeltal. Dock skall vad i punkt A 3 nedan stadgas iakttagas.

3. För Järnvägsforsens kraftverk givna föreskrifter om minim. tappning genom regleringsdammen skall uppfyllas.

4. Dygnsregleringen skall återregleras helt vid Skallböle kraftverk enligt bestämmelserna i punkt B 5.

5. Dygnsreglering får ej bedrivas när årsreglerad vattenföring vid Skallböle kraftverk överstiger 250 m³/s.

6. Vid regleringen skall tillses att i vad på dess skötsel beror gällande vattenhushållningsbestämmelser för nedströms belägna företag kan uppfyllas.

B. Bestämmelser för nedströms belägna kraftverk

1. Parteboda kraftverk (Ångesjön)

För anpassning av den vid Järnvägsforsens kraftverk bedrivna dygnsregleringen må sökandena utnyttja ett dygnsmagasin i Ångesjön mellan dämningsgränsen +158,05 m och +157,85 m.

2. Hermansboda kraftverk

För anpassning av den vid Järnvägsforsens kraftverk bedrivna dygnsregleringen må sökandena utnyttja ett dygnsmagasin uppströms kraftverksdammen vid Hermansboda kraftverk i sådan omfattning att vattenståndet vid dammen varierar mellan dämningsgränsen +123,50 m och +123,35 m.

3. <u>Ljunga kraftverk</u> (Borgsjön - Johannisbergsselet)

För anpassning av den vid Järnvägsforsens kraftverk bedrivna dygnsregleringen må sökandena utnyttja ett dygnsmagasin uppströms kraftverksdammen vid Ljunga kraftverk av högst sådan omfattning att vattenståndet vid dammen varierar mellan dämningsgränsen +112,00 m och +111,50 m. Dygnsregleringen får endast utnyttjas på sådant sätt att vattenståndet i Borgsjön på grund av regleringen aldrig överstiger +112,50 m och aldrig understiger +112,00 m och inte ger vattenståndsvariationer i sjön under dygnet större än 15 cm.

4. <u>Nederede kraftverk</u> (Torpsjön)

Under en prövotid av tre år räknat från idrifttagandet av den senaste av de båda regleringarna i Ljungan och Gimån skall gälla följande: Skillnaden mellan högsta och lägsta vattenstånd i Torpsjön får i vad på dessa korttidsregleringar beror icke överstiga 60 cm per dygn och skall under tiden 16.5-30.9 inriktas på en begränsning till högst 40 cm per dygn, allt mätt vid kontrollpegeln i Torpsjön. Därest så erfordras för att uppfylla dessa villkor berättigas och förpliktas sökandena att i erforderlig mån avsänka vattenståndet vid Nederede kraftverksdamm. Avsänkningen får dock maximalt uppgå till högst 40 cm under dämningsgränsen +58,60 m. I övrigt får någon avsänkning ej ske vid kraftverksdammen annat än i form av sådana mindre variationer som normalt kan erfordras av rent driftmässiga skäl. Dock må under prövotiden avsänkningar tillfälligtvis ske vid kraftverksdammen även då så enligt ovan icke är tillåtet, i syfte att utröna sambandet mellan avsänkning vid dammen och variationer i Torpsjön.

-.

8. KONTROLLFÖRESKRIFTER

Sökandena har föreslagit att följande bestämmelser skall gälla för kontroll av regleringarna såväl i Ljungan som i Gimån (aktbil. 399 och 703, bil. A:34 samt prot.bil. 7 till aktbil. 786 i målet A 43/61/a och aktbil. 79 i målet A 31/68):

SMHI förordnas att på sökandenas bekostnad vara kontrollant för korttidsregleringen och i denna egenskap övervaka, att de för regleringen gällande vattenhushållningsbestämmelserna iakttages. Det åligger sökandena att tillhandahålla institutet alla de uppgifter som institutet finner erforderliga för uppdragets fullgörande.

Överträder sökandena gällande vattenhushållningsbestämmelser, skall detta av kontrollanten anmälas till vattenrättsdomaren.

Sökandena har att så snart lämpligen kan ske anordna och för framtiden vidmakthålla de peglar eller skalor som SMHI finner erforderliga ur kontrollsynpunkt. Platserna för de olika anordningarna skall anvisas eller godkännas av SMHI.

Vattenstånd skall avläsas på det sätt och på de platser som SMHI med beaktande av parternas önskemål föreskriver. Reglerade och oreglerade vattenstånd och vattenföringar skall beräknas i den omfattning som SMHI föreskriver.

Sökandena skall låta anteckna och hålla tillgängliga uppgifter om vattenståndsobservationer och andra omständigheter som enligt SMHI:s uppfattning kan vara av värde för bedömning av hur regleringen utövas.

Part som önskar ändrade eller utvidgade kontrollbestämmelser kan begära detta hos SMHI.

Åtnöjes ej part med besked som SMHI lämnat i fråga rörande kontrollen av regleringarnas handhavande må frågan underställas vattendomstolens prövning.

Motparterna har godtagit de sålunda föreslagna föreskrifterna (prot.bil. 7 till aktbil. 786 i mål A 43/61 och aktbil. 79 i mål A 31/68).

ÖSV.

LJUNGANS REGLERINGAR (A 43/61)

Nedre Ljungans korttidsreglering, etapp II (VA 15/75)

Deldom meddelad den 25 januari 1983

Transumt

.-

5. HÖJDSYSTEM (VA 15/75 och A 43/61)

I domen angivna höjdsiffror hänför sig till de höjdsystem, som gäller för respektive kraftverks- och regleringsföretag utmed den aktuella delen av Ljungans vattensystem.

.-

6. TILLSTÅND TILL NEDRE LJUNGANS KORTTIDSREGLERING, ETAPP II, (VA 15/75)

.-

6.2.2. Tillstånd

Regeringen har i sitt beslut 5.7.1979 medgivit att tillstånd lämnas till etapp II med den inskränkningen att fullständig återreglering året om skall ske vid Viiforsens kraftverk. Vid detta förhållande och enär vad därefter förekommit i målet ej utgör hinder däremot, lämnar vattendomstolen sökandena tillstånd att utöva korttidsreglering i Nedre Ljungan enligt nedan angiven precisering.

.-

6.2.3.1 Särskilda villkor beträffande korttidsreglering i Holmsjön.

Vattendomstolen föreskriver, att tillståndet att utnyttja amplituderna i Holmsjön skall vara anknutet till den angivna, nuvarande utbyggnaden av Järnvägsforsens kraftverk samt att erforderlig skadereglering skall anpassas härefter.

Skulle i en framtid begäras tillstånd att avleda mera vatten genom kraftverket än 145 m3/s, varigenom det skulle bli möjligt att utnyttja de i denna dom tillstadda amplituderna i större omfattning än vad som för närvarande kan ske, skall i anslutning härtill behandlas de skador som kan uppkomma såväl i Holmsjön som nedströms kraftverket av ett sådant större utnyttjande.

6.2.3.2 Särskilda villkor beträffande korttidsregleringen i Stödesjön

Vattendomstolen föreskriver såsom särskilt villkor för tillståndet, i vad det avser Stödesjön, att sökandena till etapp II skall svara för de beräkningar, som erfordras för kontroll av att regleringsmagasinen i Gimån icke användes i högre grad än som följer av den begränsning av utnyttjandet som punkt 3 i vattenhushållningsbestämmelserna för Gimåns utvidgade korttidsreglering ger upphov till.

Sveriges Meteorologiska och Hydrologiska Institut (SMHI) förordnas att utöva kontroll över ovan angivna beräkningar och drift. Kostnaden härför skall bestridas av sökandena till etapp II.

6.2.3.3 Vattenhushållningsbestämmelser

Med ovan angiven bakgrund har vattendomstolen i nedan angivna vattenhushållningsbestämmelser sammanfattat de tillstånd till korttidsreglering som lämnats ovan och de övriga bestämmelser för vattenhushållningen som domstolen funnit skola gälla för etapp II.

Vattendomstolen fastställer följande vattenhushållningsbestämmelser för Nedre Ljungans korttidsreglering, etapp II:

A. Korttidsreglering vid Järnvägsforsens kraftverk

1. Med iakttagande av dels de för Holmsjöns årsreglering gällande bestämmelserna med det undantag som anges i punkt A 2 nedan och dels vad som i övrigt föreskrives i nedanstående vattenhushållningsbestämmelser må sökandena utöva korttidsreglering vid Järnvägsforsens kraftverk och därvid utnyttja årsregleringsmagasinet i Holmsjön i sådan omfattning, att skillnaden mellan högsta och lägsta vattenstånd i sjön i vad på korttidsregleringen beror icke överstiger 20 cm per dygn och vecka under tiden 16.5 - 30.11 och icke överstiger 20 cm per dygn och 50 cm per vecka under tiden 1.12 - 15.5.

2. Korttidsreglerad tappning skall framsläppas genom kraftstationen, varvid vattenföringen får variera mellan 0 och 145 m3/s. För Holmsjöns årsreglering föreskriven tappning enligt dom 25.6.1970 punkterna 2 och 4 får då korttidsreglering bedrives tappas som dygnsmedeltal. Dock skall vad i punkt A 3 nedan stadgas iakttagas.

3. För Järnvägsforsens kraftverk givna föreskrifter om minimitappning genom regleringsdammen skall uppfyllas.

4. Vid korttidsregleringen skall tillses att i vad på dess skötsel beror gällande vattenhushållningsbestämmelser för nedströms belägna företag kan uppfyllas.

B. Korttidsreglering vid Parteboda kraftverk

För korttidsreglering vid Parteboda kraftverk må - med iakttagande av vad som föreskrives i nedanstående vattenhushållningsbestämmelser - utnyttjas ett magasin i Ångesjön mellan dämningsgränsen + 158,05 m och + 157,85 m.

C. Korttidsreglering vid Hermansboda kraftverk

För korttidsreglering vid Hermansboda kraftverk må - med iakttagande av vad som föreskrives i nedanstående vattenhushållningsbestämmelser - utnyttjas ett magasin uppströms kraftverks-dammen i sådan omfattning att vattenståndet vid dammen varierar mellan dämningsgränsen + 123,50 m och + 123,35 m.

OSV.

B A Pricecurve

Delgives: PDP, PDER, PDN-KRD, PDN-HEÖ, PDN-NBO, PDN-MRP, PDN-NSÖ

SYDKRAFT

PROGNOSERAT GRÄNSKRAFTPRIS

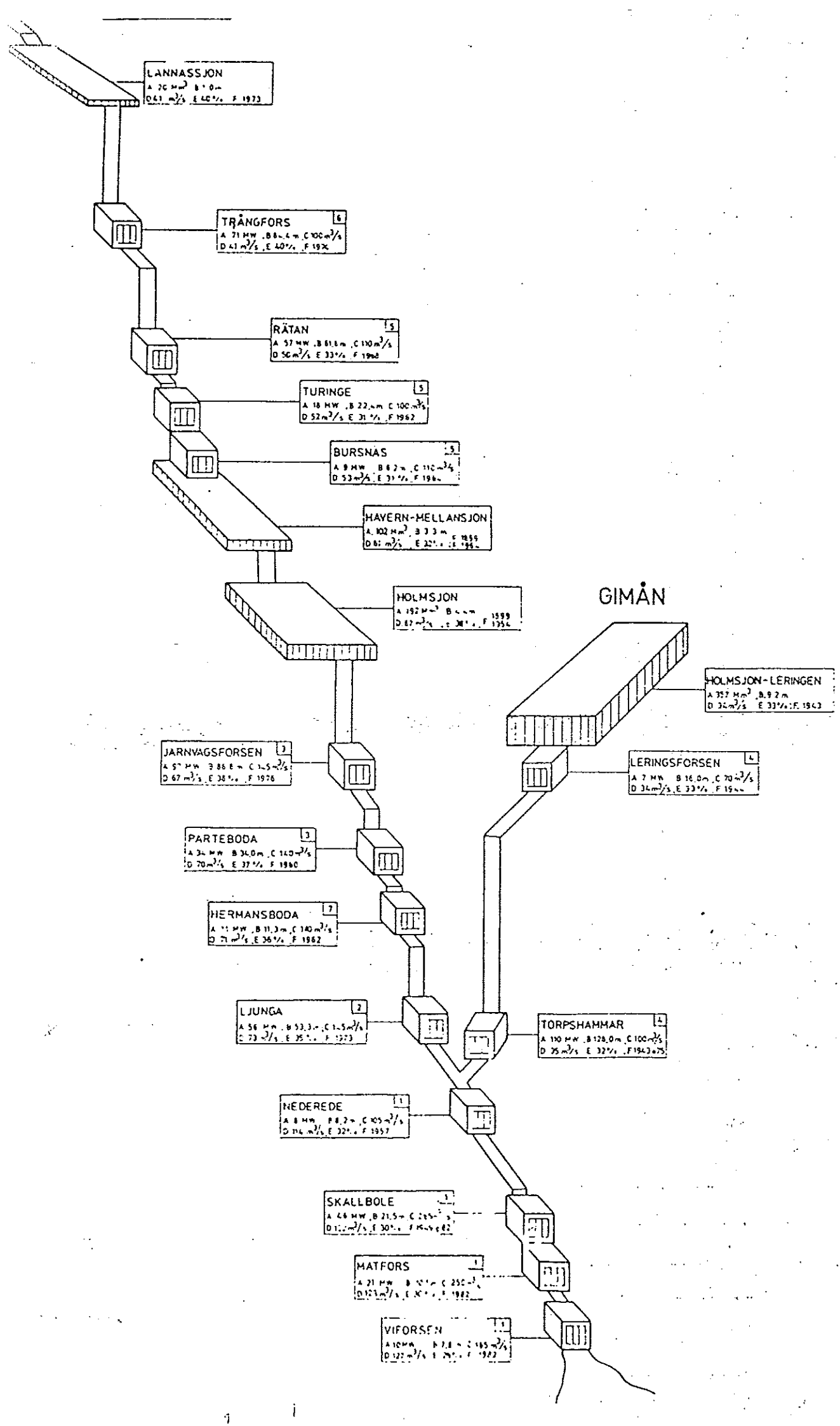
VECKA 824-25

Kl.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Gränspris kr/MWh															
	Dag/datum															
	Lö 18.6		Sö 19.6		Mö-Ons 20-22.6			Tors 23.6		Fre-Lör 24-25.6			Sön 26.6			
7–8	40		35		80			80		40			35			
8–9	45		40		|		|		45			40			
9–10	50		45		|		|		45			|			
10–11	|		|		|		|		|			40			
11–12	|		|		80			80		|			45			
12–13	50		|		70			70		|			|			
13–14	45		45		|			|		|			|			
14–15	|		40		70			70		|			|			
15–16	|		|		60			60		|			|			
16–17	|		|		50			50		|			|			
17–18	|		40		|			|		45			45			
18–19	|		45		|			50		40			40			
19–20	|		|		|			45		|			|			
20–21	|		|					|		|			|			
21–22	|		45		50				|		40			|			
22–23	45		40		45			45			35			|			
23–24	40		|		40			40			|			|			
0–1	|		|		|			|		|			|			
1–2	|		|		|			|		|			|			
2–3	|		|		|			|		|			|			
3–4	|		|		|			|		|			|			
4–5	40		40		40			|		|			40			
5–6	35		45		45			|		|			45			
6–7	35		55		55			40		35				55			
7–7																	

Heh 880614

SK 41181-3

C Map of Nedre Ljungan


LANNASSJON
TRÅNGFORS
RÄTAN
TURINGE
BURSNÄS
HAVERN-HELLANSJON
HOLMSJON
GIMÅN
HOLMSJON-LERINGEN
JÄRNVÄGSFORSEN
LERINGSFORSEN
PARTEBODA
HERMANSBODA
LJUNGA
TORPSHAMMAR
NEDEREDE
SKÄLLBOLE
MATFORS
VIFORSEN

D Program Code for the Planner

This code has been "Englified" to make it possible for an English speaking person to pronounce and understand the variable names. The code was originally written in Swedish.

```
; ************************************************************************
;
;                      ***     INITIATION     ***
;
; ************************************************************************
;
;       INIT INITIATES THE GLOBAL VARIABELS:
;                    Station              : The power station
;                    Number_of_hours      : Number of hours for which the
;                                           plan are made. Must be 168 = a week.
;                    Maximal_flow         : Maximal flow through the station.
;                    Prices               : List of the power prices for each
;                                           hour.
;                    Pricelist            : List consisting of the used pricies.
;                    Week                 : List of the hours of the week, every
;                                           hour has the properties  flow,
;                                           day, contents, and price.
;                    Average_flow         : Average water flow through the
;                                           river in km3/s.
;                    Amount_of_water      : The total flow through the river.
;                    Initial_amount_of_water: Same as Amount_of_water.
;                    Initial_content      : The reservoir contents when you
;                                           start planning.
;                    First_day            : The first day of the week (yymmdd).
;
;       INIT USES THE FUNCTIONS:
;                                      read_prices
;                                      create_pricelist
;                                      read_first_day
;                                      read_average_flow
;                                      read_station
;                                      initiate_station
;                                      read_initial_amount_of_water
;

(define (init)
        (set! Station (read_station))
        (set! Number_of_hours 168)
        (set! Average_flow (read_average_flow))
        (set! Initial_content (read_initial_amount_of_water))
        (set! First_day (read_first_day))
        (initiate_station Station)
        (set! Maximal_flow (getprop station 'maximalflow))
        (set! Max_content (getprop station 'max_content))
        (set! Prices (read_prices))
        (set! Pricelist (create_pricelist Prices))
        (init_week First_day Prices)
        (set! Amount_of_water (* Number_of_hours Average_flow))
        (set! Initial_amount_of_water Amount_of_water)
)


; ***** READ_STATION *****************************************************
;
;       RETURNS STATION
;       READ FROM SYS$INPUT

(define (read_station )
   (let (( station "" ))
        ( do (( i 0 i))
             ((not (eqv? station "")) station)
             ( display "Which station?" )
             ( set! station (read) )
        )
   )
)

; ***** END OF READ_STATION **********************************************
```

```
; ***** READ_AVERAGE_FLOW ***************************************************
;
;       RETURNS THE AVERAGE WATER FLOW, READ FROM SYS$INPUT

(define (read_average_flow)
   (let (( flow 0 ))
        ( do (( flow 0 flow))
             ((and (number? flow) (positive? flow))
                  flow
             )
             ( display "What is the average flow?")
             ( set! flow (read) )
        )
   )
)

; ***** END OF READ_AVERAGE_FLOW  ******************************************



; ***** READ_INITIAL_AMOUNT_OF_WATER ***************************************
;
;       RETURNS THE INITIAL AMOUNT OF WATER IN THE MAGASINE
;       THIS IS READ FROM SYS$INPUT

(define (read_initial_amount_of_water)
   (let (( contents -1 ))
        ( do (( contents -1 contents))
             ((and (number? contents) (>=  contents 0))
                  contents
             )
             ( display "What is the initial amount of water?")
             ( set! contents (read) )
        )
   )
)

; ***** END OF READ_INITIAL_AMOUNT_OF_WATER  ***************************************



; ***** READ_FIRST_DAY ***************************************************
;
;       RETURNS THE FIRST DAY OF THE WEEK ON THE FORM YYMMDD
;       FROM SYS$INPUT

(define (read_first_day)
   (let (( day 0 ))
        ( do (( day 0 day))
             ((and (number? day) (positive? day))
                  day
             )
             ( display "What is the first day of the week (yymmdd)?")
             ( set! day (read) )
        )
   )
)

; ***** END OF READ_FIRST_DAY  *********************************************


; ***** INITIATE_STATION ***************************************************
;

(define (initiate_station station)
        (case station
            ( (jfn jaernvaegsforsen jarnvagsforsen jaernvaegs_forsen
```

```
                jarnvags_forsen
               ) (initiate_station_jfn))
        )
)

(define (initiate_station_jfn)
        (putprop 'jfn 'flows (list 65 65))
        (putprop 'jfn 'maximalflow 120)
        (putprop 'jfn 'tillrinning (make-list Number_of_hours Average_flow))
        (putprop 'jfn 'minimalflows (make-list Number_of_hours 0))
        (putprop 'jfn 'max_content 600)
        (putprop 'jfn 'torpshammarflows (make-list Number_of_hours 0))
)

; **** END OF INITIATE_STATION ************************************************


; ****  READ_PRICES ***********************************************
;
;       RETURNS A GRAENSKRAFTS PRIS KURVA READ FROM
;       pricelist.dat
;       CALLS THE FUNCTIONS   read_prices_1
;                             read_pricefile

(define (read_prices)
  (read-char)
  (let ((file (read_pricefile)))
    (let (( port (open-input-file file)))
         (let(( l (read_prices_1 port)))
             ( close-input-port port)
            l
         )
    )
  )
)

(define (read_prices_1 port)
        ( let (( x (read port)))
              (cond (( eof-object? x) ())
                    ( #t (append! (list x) (read_prices_1 port)))
              )
        )
)

(define (read_pricefile)
        (display "In which file is the pricelist kept? (pricelist.dat)")
        (let ((ch (read-char))(str ""))
           (cond
              ((char=? ch #\newline)
                "pricelist.dat"
              )
              ((char=? ch #\")
                (unread-char ch)
                (read)
              )
              (#t
                (clear-input-port)
                (read_pricefile)
              )
           )
        )
)

; ***** END OF READ_PRICES ****************************************


; ***** CREATE_PRICELIST ******************************************************
;
;       RETURNS A LIST WHERE THE ELEMENTS OF PRICELIST APPEAR ONCE FOR
```

```
;       EACH KIND OF ELEMENT
;       CALLS THE FUNCTION create_pricelist_1

(define (create_pricelist pricelist)
   (create_pricelist_1 (sort > (list-copy pricelist)))
)

(define ( create_pricelist_1 pricelist)
   ( cond
       ((null? pricelist) '())
       ((= 1 ( length pricelist)) (list (car pricelist)))
       (( eqv? ( car pricelist) (cadr pricelist))
          ( create_pricelist_1 ( cdr pricelist))
       )
       ( #t (cons (car pricelist)
                  (create_pricelist_1 (cdr pricelist))
       ))
   )
)

; ***** END OF CREATE_PRICELIST ************************************************



; ***** INIT_WEEK ***************************************************************
;
;       CREATES THE GLOBAL LIST WEEK AND GIVES EVERY HOUER THE
;       PROPERTIES flow, price, day, contents
;
;       INIT_WEEK CALLS THE FUNCTION init_week_1

(define ( init_week first_day pricelist)
        (set! Week  (let ((i 0) (v (make-list Number_of_hours 'hour))
                          (day (remainder first_day 10000))
                          (year (quotient first_day 10000))
                     )
                     (init_week_1 day year pricelist i)
                  )
        )
)

(define ( init_week_1 day year pricelist i)
        (cond
            ((= i 128) ())
            ((= 1 (length pricelist))
                (let ((post (gensym)))
                     (putprop post 'flow 0)
                     (putprop post 'day day)
                     (putprop post 'price (car pricelist))
                     (putprop post 'contents 0)
                     (list post)
                )
            )
            (#t (1+ i)
                (let ((post (gensym)))
                     (putprop post 'flow 0)
                     (putprop post 'day day)
                     (putprop post 'price (car pricelist))
                     (putprop post 'contents 0)
                     (cons post
                           (init_week_1 (if (= 0 (remainder 1 24))
                                  (datum+ day year) day) year (cdr pricelist) i
                           )
                     )
                )
            )
        )
)
```

```
; ***** END OF INIT_WEEK ************************************************************

; ***** DATUM+ ***********************************************************************
;
;       INCREMENTS day IN AN INTELIGENT WAY
;

(define (datum+ day year)
        (case day
            (0131 0201)
            (0228 (if (= 0 (remainder year 4)) 0229 0301))
            (0229 0301)
            (0331 0401)
            (0430 0501)
            (0531 0601)
            (0630 0701)
            (0731 0801)
            (0831 0901)
            (0930 1001)
            (1031 1101)
            (1130 1201)
            (1231 0101)
        (else (1+ day))
        )
)

; ***** END OF DATUM+ ****************************************************************


; ***********************************************************************************

;                         ***    END OF INIT    ***

; ***********************************************************************************



; ***** PRINT_FLOWS ******************************************************
;
;       WRITES WEEK.CONTENTS AND WEEK.FLOW IN A HANDSOME WAY
;       TO SYS$OUTPUT

(define (print_flows week)
   (let ((contentsvect (make-vector 7 0))(flowvect (make-vector 7 0))
         (pricevect (make-vector 7 0)))
        (newline)
        (display "     day 1          day 2          day 3          day 4          da
        (newline)
        (display " con flw price   con flw price   con flw price   con flw price   co
        (newline)
        (newline)
        (do ((i 0 (1+ i)))
            ((= i 24))
            (do ((j 0 (1+ j)))
                ((= j 7) (newline))
                (let (( hour (list-ref week (+ i (* j 24)))))
                     (let ((contents (getprop hour 'contents))
                           (flow   (getprop hour 'flow))
                           (price (getprop hour 'price))
                     )
                     (vector-set! contentsvect j (+ (vector-ref contentsvect j)
                         contents)
                     )
                     (vector-set! flowvect j (+ (vector-ref flowvect j)
                         flow)
                     )
```

```
                        ( case (floor (* (log (abs contents)) 0.434294482)) ;LOG 10
                            ( 0 (display (format "   ~a" contents)))
                            ( 1 (display (format "  ~a" contents)))
                            ( 2 (display (format " ~a" contents)))
                            ( 3 (display (format "~a" contents)))
                            ( 4 (display (format "~a" contents)))
                        )
                        ( case (floor (* (log flow) 0.434294482)) ;LOG 10
                            ( 0 (display (format "   ~a " flow)))
                            ( 1 (display (format "  ~a " flow)))
                            ( 2 (display (format " ~a " flow)))
                            ( 3 (display (format "~a " flow)))
                            ( 4 (display (format "~a " flow)))
                            ( 5 (display (format "~a " flow)))
                        )
                        ( case (floor (* (log price) 0.434294482)) ;LOG 10
                            ( 0 (display (format "   ~a  " price)))
                            ( 1 (display (format "  ~a  " price)))
                            ( 2 (display (format " ~a  " price)))
                            ( 3 (display (format "~a  " price)))
                            ( 4 (display (format "~a  " price)))
                            ( 5 (display (format "~a  " price)))
                        )
                    )
                )
            )
        )
        (newline)
        (do ((i 0 (1+ i)))
            ((= i 7))
            (let ((contents (quotient (vector-ref contentsvect i) 24))
                  (flow (quotient (vector-ref flowvect i) 24))
                )
                ( case (floor (* (log (abs contents)) 0.434294482)) ;LOG 10
                    ( 0 (display (format "   ~a" contents)))
                    ( 1 (display (format "  ~a" contents)))
                    ( 2 (display (format " ~a" contents)))
                    ( 3 (display (format "~a" contents)))
                    ( 4 (display (format "~a" contents)))
                )
                ( case (floor (* (log flow) 0.434294482)) ;LOG 10
                    ( 0 (display (format "   ~a       " flow)))
                    ( 1 (display (format "  ~a       " flow)))
                    ( 2 (display (format " ~a       " flow)))
                    ( 3 (display (format "~a       " flow)))
                    ( 4 (display (format "~a       " flow)))
                    ( 5 (display (format "~a       " flow)))
                )
            )
        )
    )
    (newline)
)

; ***** END OF PRINT_FLOWS ************************************************


; ***** PLACE_EVENLY ******************************************************
;
;
;       FOR TIME:= 1 TO 24 DO
;         FOR DAY:= 1 TO 7 DO
;           IF HOUR.PRICE= price AND Amount_of_water > x * Initial_amount_of_water
;           THEN
;             HOUR.FLOW := MAXIMALFLOW
;
;       CALLS THE FUNCTION place_hour
```

```
(define (place_evenly Week  price x)
        (let ((mon ())(tue ())(wed ())(thu ())(fri ())(sat ())(sun ()))
             (let work ((i 0)(Head (car Week ))(Tail (cdr Week )))
                  (cond
                      ((= i (1- Number_of_hours))
                         (set! sun (append! sun (list Head)))
                      )
                      (#t (case (quotient i 24)
                                (0 (set! mon (append! mon (list Head))))
                                (1 (set! tue (append! tue (list Head))))
                                (2 (set! wed (append! wed (list Head))))
                                (3 (set! thu (append! thu (list Head))))
                                (4 (set! fri (append! fri (list Head))))
                                (5 (set! sat (append! sat (list Head))))
                                (else (set! sun (append! sun (list Head))))
                          )
                          (work (1+ i)(car Tail)(cdr Tail))
                      )
                  )
             )
             (map place_hour mon tue wed thu fri sat sun
                 (make-list 24 price)(make-list 24 x)
             )
        )
)


(define (place_hour hour1 hour2 hour3 hour4 hour5 hour6 hour7
         price x)
        (let (( l (list hour1 hour2 hour3 hour4 hour5 hour6 hour7))
              (waterlimit (* x Initial_amount_of_water))
             )
             (let place_hour_1 (( H (car l)) (T (cdr l)))
                  (cond
                      ((= price (getprop H 'price))
                         (cond
                              ((> Amount_of_water waterlimit)
                                  (putprop H 'flow Maximal_flow)
                                  (set! Amount_of_water (- Amount_of_water Maximal_flo
                                  ( if T (place_hour_1 (car T) (cdr T)))
                              )
                         )
                      )
                      (T (place_hour_1 (car T) (cdr T)))
                  )
             )
         )
)

; ***** END OF PLACE_EVENLY ************************************************



; ******************************************************************************
;
;                           ***     CLEAN_UP    ***
;
; ******************************************************************************
;
;        TAKES THE RULES IN rulelist AND APPLIES THESE TO WEEK.
;        TO INTRODUCE A NEW CLENING UP RULE JUST WRITE IT IN RULELIST
;
;

(define (clean_up station)
        (let ((rulelist (list clean_up_1 clean_up_start_and_stop)))
```

```
            (let work (( Head (car rulelist))(Tail (cdr rulelist)))
                 (Head station)
                 (if Tail (work (car Tail)(cdr Tail)))
            )
        )
)

; ************************************************************************

;                    ***    END OF CLEAN_UP    ***

; ************************************************************************



; ***** CLEAN_UP_1 *********************************************************
;
;       IF THE TIME BETWEEN TWO HOURS WITH Maximal_flow IS ONE HOUR THAT
;       HOUR IS GIVEN THE FLOW Maximal_flow.
;

(define (clean_up_1 station )
        (let work ((v Week ))
             (cond
                 ((null? (cddr v)) ())
                 (#t (cond
                         ((and (= Maximal_flow (getprop (car v) 'flow)
                                   (getprop (caddr v) 'flow))
                               (<  (getprop (cadr v) 'flow) Maximal_flow))
                            (set! Amount_of_water (- Amount_of_water Maximal_flow))
                            (putprop (cadr v) 'flow Maximal_flow)
                            (putprop (cadr v) 'flag 'haal)
                         )
                     )
                     (work (cdr v))
                 )
             )
         )
)

; ***** END OF CLEAN_UP_1 *************************************************



; ***** CLEAN_UP_START_AND_STOP ********************************************
;
;       STARTS AND STOPS ONE UNIT PER 30 MINUTES
;
;       CALLS THE FUNCTIONS:        clean_up_2
;                                   clean_up_4
;                                   running?
;                                   stoped?
;                                   create_startflows
;

(define (clean_up_start_and_stop station)
        (clean_up_2 station)
        (clean_up_4 station)
        (clean_up_5 station)
)

; ***** CLEAN_UP_2 *********************************************************
;
;       REPEAT
;         K:= K+1
;         IF (STOPED x HOURS) AND (RUNNING CEILING( NUMBER_OF_UNITS/ 2)
;               HOURS THEREAFTER) THEN
```

```
;           FOR J:= 1 TO CEILING( NUMBER_OF_UNITS/ 2) DO
;             BEGIN
;               WEEK[K].FLOW:= STARTFLOWS[J]
;               WEEK[K].FLAG:= START
;             END
;       UNTIL FOUR ELEMENTS LEFT IN WEEK
;
;       CALLS:    stoped?
;                 running?
;                 create_startflows

(define (clean_up_2 station)
        (let ((starttime (ceiling (/ (length (getprop station 'flows)) 2)))
              (startflows (create_startflows station))
              (x 1)
             )
             (let work ((v Week ))
                  (cond
                      ((and (stoped? v 0 starttime)(running? v starttime x)
                            (not (eqv? (getprop (car v) 'start) 'stop))
                       )
                         (work
                             (let place_startflows ((v1 v)(s startflows))
                                  (cond
                                      ((= (length s) 1)
                                         (putprop (car v1) 'flow (car s))
                                         (putprop (car v1) 'flag 'start)
                                         v1
                                      )
                                      (#t (putprop (car v1) 'flow (car s))
                                          (putprop (car v1) 'flag 'start)
                                          (place_startflows (cdr v1)
                                               (cdr s))
                                      )
                                  )
                             )
                         )
                      ((null? (cdr v)) ())
                      ((null? (cddr v)) ())
                      ((null? (cdddr v)) ())
                      ((null? (cddddr v)) ())
                      (#t (work (cdr v)))
                  )
             )
        )
)

; ***** END OF CLEAN_UP_2 ************************************************

; ***** CLEAN_UP_4 ********************************************************
;
;       REPEAT
;         K:= K+1
;         IF (RUNNING x HOURS) AND (STOPED CEILING( NUMBER_OF_UNITS/ 2)
;               HOURS THEREAFTER) THEN
;           BEGIN
;             K:=K+x
;             FOR J:= 1 TO CEILING( NUMBER_OF_UNITS/ 2) DO
;               BEGIN
;                 WEEK[K].FLOW:= STARTFLOWS[J]
;                 WEEK[K].FLAG:= START
;               END
;           END
;       UNTIL FOUR ELEMENTS LEFT IN WEEK
;
;       CALLS:    stoped?
;                 running?
;                 create_startflows
```

```
(define (clean_up_4 station)
        (let (( stoptime (ceiling (/ (length (getprop station 'flows)) 2)))
              (stopflows (reverse (create_startflows station)))
              (x 1)
             )
             (let work ((v Week ))
                  (cond
                      ((and (running? v 0 x)(stoped? v x stoptime)
                            (not (eqv? (getprop (car v) 'flag) 'stop))
                       )
                          (work
                             (let place_stopflow ((v1 (list-tail v
                                                          x))(s stopflows)
                                                  )
                                  (cond
                                      ((= (length s) 1)
                                         (putprop (car v1) 'flow (car s))
                                         (putprop (car v1) 'flag 'stop)
                                         v1
                                      )
                                      (#t (putprop (car v1) 'flow (car s))
                                          (putprop (car v1) 'flag 'stop)
                                          (place_stopflow (cdr v1)
                                               (cdr s))
                                      )
                                  )
                             )
                          )
                      )
                      ((null? (cdr v)) ())
                      ((null? (cddr v)) ())
                      ((null? (cdddr v)) ())
                      ((null? (cddddr v)) ())
                      (#t (work (cdr v)))
                  )
             )
        )
)

; ***** END OF CLEAN_UP_4 ************************************************


; ***** CLEAN_UP_5 ********************************************************
;
;       IF week[1].flow >0 THEN week[1].flag := start
;       IF week[Number_of_hours].flow >0 THEN
;          week[Number_of_hours].flag := start

(define (clean_up_5 station)
        (if (> (getprop (car Week ) 'flow) 0) (putprop (car Week ) 'flag
           'start)
        )
        (if (> (getprop (car (last-pair Week )) 'flow) 0)
               (putprop (car (last-pair Week )) 'flag 'stop)
        )
)

; ***** END OF CLEAN_UP_5 ************************************************


; ***** STOPED? ***********************************************************
;
;       TRUE IF FLOW=0 THE HOURS x TO x+ y

(define (stoped? week x y)
        (cond
            (( null? week) #f)
            ((> x 0) (stoped? (cdr week) (1- x) y))
```

```
            ((zero? (getprop (car week) 'flow))
             (if (= y 1) #t (stoped? (cdr week) 0 (1- y)))
            )
            (#t #f)
         )
)

; ***** END OF STOPED ************************************************************


; ***** RUNNING? *****************************************************************
;
;       TRUE IF FLOW<>0 THE HOURS x TO x+ y

(define (running? week x y)
        (cond
            ((null? week) #f)
            ((> x 0) (running? (cdr week) (1- x) y))
            ((positive? (getprop (car week) 'flow))
               (if (= y 1) #t (running? (cdr week) 0 (1- y)))
            )
            (#t #f)
        )
)

; ***** END OF RUNNING? **********************************************************


; ***** CREATE_STARTFLOWS *********************************************************
;
;       RETURNS A LIST THERE EVERY ELEMENT GIVES THE FLOW NEEDED TO START
;       ONE UNIT IN station EVERY HALF HOUR
;

(define (create_startflows station)
        (let work (( l (sort < (getprop station 'flows)))
                   (number (length (getprop station 'flows)))
                   (earlier 0)
                  )
                  (cond
                      ((= 1 number) ())
                      ((even? number)
                          (cons (quotient (car l) 2)
                              (work (cdr l)(1- number)(car l))
                          )
                      )
                      (#t (cons (+ (car l)(quotient (cadr l) 2) earlier)
                                (work (cddr l) (- number 2)
                                    (+ (car l) (cadr l) earlier)
                                )
                          )
                      )
                  )
        )
)

; ***** END OF CREATE_STARTFLOWS **************************************************


; ********************************************************************************

;               *****     END OF CLEAN_UP_START_AND_STOP *****

; ********************************************************************************
```

```
; ***** MAINPROGRAM ***********************************************************
;

(define (mainprogram)
        (do
           ((str "j" str))
           ((not (member str (list "j" "J" "y" "Y" "ja" "Ja"
                                   "JA" "yes" "Yes" "YES")
                 )
           ))
           (init)
           (place_mimflows station)
           (let work ((price (car pricelist)) (list_a (cdr pricelist))
                      (x (- 1 (read_x)))
                 )
                (cond
                    ((null? list_a) ())
                    (( > Amount_of_water (* x Initial_amount_of_water))
                       (place_evenly Week  price x)
                       (work (car list_a) (cdr list_a) x)
                    )
                )
           )
           (clean_up Station)
           (apply_reservoir_rules Week )
           (print_flows Week )
           (display "One more time?")
           (set! str (symbol->string (read)))
           (newline)
        )
        "FORTRAN STOP"
)

(define (read_x)
        ( do (( x 0 x))
             ((and (number? x) (positive? x))
                 x
             )
      ( display "How big part of the water should be placed?")
             ( set! x (read) )
        )
)

; ***** END OF MAINPROGRAM ***************************************************


; ***** PLACE_MIMFLOWS ********************************************************
;
;       FOR I:= 1 TO Number_of_hours DO
;         Week [I].flow := station.minimalflows[I]
;

(define (place_mimflows station)
        (let work (( v Week ) (minlist (getprop station 'minimalflows)))
             (cond
                 ((null? v) ())
                 (#t (putprop (car v) 'flow (car minlist))
                     (work (cdr v) (cdr minlist))
                 )
             )
        )
)

; ***** END OF PLACE_MIMFLOWS ************************************************


; ***** UPDATE_CONTENT ********************************************************
```

```
;
;       FOR I:= 1 TO Number_of_hours DO
;         Week [I].contents := Week [I-1].contents+ Average_flow- Week [I].flow
;

(define (update_content list_a)
        (let work ((contents Initial_content)(l list_a))
             (cond
               ((null? l) contents)
               (#t
                    (let ((s (-(+ contents Average_flow)(getprop (car l) 'flow))))
                         (putprop (car l) 'contents s)
                         (work s (cdr l))
                    )
               )
             )
        )
)

; ***** END OF UPDATE_CONTENT ************************************************



; ***** APPLY_RESERVOIR_RULES ************************************************

;       UPPDATES CONTENT AND ADJUSTS FLOW IN ACORDANCE TO THE RESERVOIR
;       RULES
;
;       CALLS:   update_content
;                work_between_blocks

(define (apply_reservoir_rules list_a)
        (update_content Week )
        (update_content Week )
        (work_between_blocks list_a)
)

; ***** END OF APPLY_RESERVOIR_RULES *****************************************


; ***** WORK_BETWEEN_BLOCKS **************************************************

;       CHECKS THAT THE RESERVOIR NOT BECAMES TO BIG BETWEEN THE MAIN RUNS
;
;       CALLS:   water_left
;                flow<>zero
;                place_best_side
;                work_on_blocks
;                place_costliest

(define (work_between_blocks list_a)
        (let ((before ())(now (car list_a))(after (cdr list_a))
                  (water 0)(i 0)
             )
             (do ((i 0))
                 ((or (> (getprop now 'contents) Max_content)
                      (null? after)
                      (eqv? (getprop now 'flag) 'start)
                 )())
                 (set! before (cons now before))
                 (set! now (car after))
                 (set! after (cdr after))
             )
             (cond
               ((null? after) ())
               ((> (getprop now 'contents) Max_content)
                 (set! water (water_left (cons now after)))
                 (set! i (flow<>zero list_a))
                 (cond
```

```
                ((and (> i 0) (< i (length before)))
                  (place_best_side i list_a water)
                  (set! now (car (list-tail list_a  i)))
                  (set! after (cdr (list-tail list_a i)))
                  (if (not (null? after)) (work_on_blocks (cons now after)))
                )
                (#t
                  (set! i (place_costliest list_a before water))
                  (place_best_side i list_a (- water
                                               (car (getprop Station 'flows)))
                  )
                  (set! now (car (list-tail list_a i)))
                  (set! after (cdr (list-tail list_a i)))
                  (if (not (null? after)) (work_on_blocks (cons now after)))
                )
              )
            )
            (#t
              (work_on_blocks (cons now after))
            )
          )
        )
)

; ***** END OF WORK_BETWEEN_BLOCKS ************************************************


; ***** WATER_LEFT ************************************************************
;
;       RETURNS THE AMOUNT OF WATER THAT MUST FLOW THROUGH THE STATION
;       UP TO THE FIRST START TO ENSURE THAT THE RESERVOIR SHOULD BE
;       FULL ATT THE MOMENT OF START.
;

(define (water_left list_a)
        (let work (( water 0)(l list_a))
             (cond
               ((null? l) water)
               ((eqv? (getprop (car l) 'flag) 'start) water)
               (#t
                 (work (+ water (- Average_flow (getprop (car l) 'flow))
```

E Program Code for the Examiner

This code has been "Englified" to make it possible for an English speaking person to pronounce and understand the variable names. The code was originally written in Swedish.

```
/* The C-Prolog 1.5 interpretator is started with the comand
   prolog -g 1000 -l 1900 -t 500 */


/* Representation of a unit

        unit(name, flow).    */

        unit(jfn, 65).
        unit(jfn,120).
        unit(jfn,0).
        unit(jfn,145).
        unit(lju,0).
        unit(lju,55).
        unit(ljujfn,110).
        unit(ljujfn,145).


/* Representation of a station

        station(name, Flow,beauty_water, day_number):-
                unit(name, Flow).*/

        station(jfn, Flow,3, _):- unit(jfn, Flow).

        station(lju, Flow, Beauty, Day):-
                unit(lju, Flow),
                (Day>= 136, Day=<243, Beauty= 0.5);
                (Beauty=0).


/* Representation of number of starts each day

        number_of_starts(name,daynumber,number)   */



increase_number_of_starts(Name,Day):- number_of_starts(Name,Day,X),
                                retract(number_of_starts(Name,Day,X)),
                                X1 is X+1,
                                Y=.. [number_of_starts, Name, Day, X1],
                                asserta(Y).

/* Representation of allowed flow in a ceartain river at a ceartain time.
        allowed(name, Earlier_flows, Flow, Hour, Day):-  .....    */

        allowed(jfn,Earlier_flows, Flow, Hour,Day):-
                write(Day),
                write('     '),
                write(Hour),
                write('     '),
                write(Flow),
                write('         '),
                contents_below_limit(jfn, Earlier_flows, Flow, Day),
                maximal_amplitude_variation(jfn, Earlier_flows, Flow, Hour),
                no_more_than_two_starts(jfn, Earlier_flows, Flow,
                Day).


        contents_below_limit(Stn, Earlier_flows, Flow, Day):-
                week_plan(Averageflow),
                count_content(Earlier_flows,Contents, Averageflow),
                writeln(Contents),
                max_content(Stn, Day, Max_content),
                to_big_content(Contents,Flow,Max_content, Averageflow).

        to_big_content(Contents,Flow,Max_content, Averageflow) :-
                Contents- Flow+ Averageflow =< Max_content.
```

```
to_big_content(Averagecontents,Flow, Max_content,_):-
        writeln('To large content').


abs(X,X):- X>= 0.
abs(X,Y):- Y is -1* X.


max_content(jfn, Day, Max_content):- Day< 136, Max_content= 1200.
max_content(jfn, Day, Max_content):- Day=< 273, Max_content= 600.
max_content(jfn, Day, Max_content):- Max_content= 1200.


count_content([],0,_).

count_content([H|T], Contents, Averageflow):-
        count_content(T, Contents_1, Averageflow),
        Contents is (Contents_1 - H+ Averageflow).


maximal_amplitude_variation(jfn, Earlier_flows, 0, Hour):-
        flow_in_thr(Flow_thr, Hour),
        amplitude_rule_OK_thr(Flow_thr, Earlier_flows,
        1).

maximal_amplitude_variation(jfn, Earlier_flows, 0, Hour):-
        writeln('If the flow through thr + jfn >= 220 '),
        writeln('for more than ten hours'),
        writeln('then Jfn must not be stoped'),
        writeln('for more than 7 hours thereafter.').

maximal_amplitude_variation(jfn, _,X, _):- X=\=0.


flow_in_thr(Flow_thr, Hour):-
        flow(thr, Flow_list),
        count_thr(Flow_list, Flow_thr, X, Hour).


count_thr([],[],0,_).

count_thr([H|T], L, X, Hour):-
        count_thr(T, L1, X1, Hour),
        X is X1+ 1,
        count1_thr(X,Hour,L,H,L1).

count1_thr(X, Hour,L,H,L1):-
        X=< Hour,
        L= [H|L1].

count1_thr(X, Hour,L,H,L1):-
        X> Hour,
        L= L1.



amplitude_rule_OK_thr([H|T], [H1|T1], X):-
        H1 =\=0,
        X=< 7.

amplitude_rule_OK_thr([H|T], [H1|T1], X):-
        X=< 7,
        H1=0,
        X1 is X+1,
        amplitude_rule_OK_thr(T, T1, X1).

 amplitude_rule_OK_thr([H|T], [H1|T1], X):-
        X>7,
```

```
        X=< 17,
        H+ H1< 200.

amplitude_rule_OK_thr([H|T], [H1|T1], X):-
        X> 7,
        X=< 17,
        H+ H1>= 200,
        X1 is  X+ 1,
        amplitude_rule_OK_thr(T, T1, X1).

amplitude_rule_OK_thr([],_,_).

amplitude_rule_OK_thr(_, [],_).


no_more_than_two_starts(Name, [],_,_).

no_more_than_two_starts(Name, [H|T], Flow,_):- H=\=0.

no_more_than_two_starts(Name, [H|T], Flow,_):- H=0, Flow=0.

no_more_than_two_starts(Name, [H|T], Flow, Day):-
        H=0,
        Flow=\=0,
        number_of_starts(Name, Day, X),
        less_than_two_starts(Name, Day,X).


less_than_two_starts(Name, Day, X):-
        X< 2,
        increase_number_of_starts(Name, Day).

less_than_two_starts(Name, _,_):-
        writeln('The unit should not be started'),
        writeln('more than two times each day.').


examine_week(Name):-
        init(Name, First_day,Flows,New_flows),
        writeln('Day  Hour  Flow     Contents'),
        Last_day is First_day+ 7,
        examine_day(Name, First_day, Last_day,
        Flows,New_flows),
        finish(Name, First_day, Flows).

examine_day(Name, Last_day, Last_day, Flows,New_flows):-
        examine_weekend(Name, Last_day, Flows).

examine_day(Name, Day, Last_day, Flows,New_flows):-
        examine_hour(Name, Day, Flows, 1,New_flows),
        Next_day is Day + 1,
        transfer24(Flows1,New_flows1,Flows,New_flows),
        examine_day(Name, Next_day, Last_day, Flows1,New_flows1).


examine_hour(_,_,_,25,_).

examine_hour(Name,Day, Flows, Hour,New_flows):-
        car(Flow, New_flows),
        allowed(Name, Flows, Flow, Hour, Day),
        Next_hour is Hour + 1,
        Flows1 = [Flow|Flows],
        cldr(New_flows1, New_flows),
        examine_hour(Name, Day, Flows1, Next_hour,New_flows1).


init(Name, First_day, Flows, New_flows):-
        write('Inputfile: '),
        read(Inputfile),
```

```
        nl,
        see(Inputfile),
        read(First_day),
        read(Weekly_plan),
        read(Flows),
        read(New_flows),
        Y=..[week_plan, Weekly_plan],
        assert(Y),
        create_number_of_starts(Name, First_day, 0).


create_number_of_starts(_,_,7).

create_number_of_starts(Name, Day, X):-
        Y is Day+ X,
        Z=.. [number_of_starts,Name, Y, 0],
        assert(Z),
        X1 is X+ 1,
        create_number_of_starts(Name, Day, X1).


finish(Name, First_day, Flows):-
        retract(week_plan(_)),
        remove_number_of_starts(Name, First_day),
        close('inputfile.dat'),
        Y=.. [flow, Name, Flows],
        assert(Y).


remove_number_of_starts(Name, First_day):-
        retract(number_of_starts(Name,_,_)),
        fail.

remove_number_of_starts(_,_).


read_list(_,3).

read_list([H|T], X):- X1 is X+ 1,
                      read_list(T,X1),
                      read(H).



writeln(X):- write(X),
             nl.

car(X,[]).
car(H,[H|T]).


cldr(0,[]).
cldr(T,[H|T]).


transfer24(Before1,After1,Before,After):-
        transfer24_1(Before1,After1,Before,After,1).

transfer24_1(Before,After,Before,After,25).

transfer24_1(Before1,After1,Before,After,N):-
        N1 is N + 1,
        transfer24_1(Before2,After2,Before,After,N1),
        car(X,After2),
        Before1 = [X| Before2],
        cldr(After1,After2).
```

```
        examine_weekend(_,_,_) :-
                writeln('Examiner_weekend is not'),
                writeln('is not implemented but can be used'),
                writeln('for checking rules that'),
                writeln('are valid for weekends.').
/* Test data */
flow(thr,[0,0,0,0,0,101,105,110,111,102,103,104,105,116,117,118,102,102,102]).
```

F User Manual for the Planner

Below will be described how to run the Planner, an expert program for planning of waterflows through powerstations and their reservoirs, written by Olof Wickström as part of his Master Thesis in the summer and autumn of 1988 at the Department of Automatic Control, Lund Institute of Technology and Sydkraft AB.

If you are using a VAX 780 under VMS version 4.5 and Chez Scheme version 2.0

Move to the directory where you keep the file:

```
expert.ss
```

Start the scheme interpretator by the command

```
scheme
```

The computer will now welcome you to

```
Chez Scheme version 2.0  Copyright (c) R. Kent Dybvig
```

and will show the prompt

```
>
```

You will now load the program by writing

```
(load "expert.ss")
```

Now everything is set to start the program by writing

```
(mainprogram)
```

The program will respond by asking what station you would like to plan.

```
Which station?
```

At the time of writing only the answer

```
jfn
```

is valid as it is the only station for which rules have been implemented.

The program will respond by asking for the average water flow.

```
What is the average flow?
```

You answer with a positive number.

The program will respond by asking for the content of the reservoir at the start of the week that should be planned.

```
What is the initial amount of water?
```

You answer with a positive number.

The program will now ask you for the first day of the week that should be planned.

```
Which is the first day of the week (yymmdd) ?
```

You answer in the form yymmdd.

Now the program will ask for the file in which the pricecurve is kept.

```
In which file is the pricelist kept? (pricelist.dat)
```

Your answer should be of the form " VMS_filename " and do not forget the double quote (") signs.

The file should bee a standard ASCI file containing 168 numbers representating the price for each hour. An typical example is:

```
0 0 0 0 0 0 0 90 90 90 90 90 50 90 90 90 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 90 90 90 90 90 50 90 90 90 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 90 90 90 90 90 50 90 90 90 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 90 90 90 90 90 50 90 90 90 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 90 90 90 90 90 50 90 90 90 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 90 90 90 90 90 50 90 90 90 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 90 90 90 90 90 50 90 90 90 0 0 0 0 0 0 0 0
```

After a little while the program asks for how much of the available water that should be placed on the most costly hours.

```
How big part of the water should be placed?
```

The available water is defined as the average water flow times seven * twentyfour (number of days times number of hours).

Your answer should be a positive number preferably around 1.0.

The program will after a while show you its plan. After this it will ask you if you want a second run.

```
Once more?
```

Your answer would be

```
yes
```

or

```
no
```

If you answer yes you will be asked for a station and if you answer no the reply from the machine will be

```
>
```

If you now want to leave Scheme you write

```
(exit)
```

G User Manual for the Examiner

Below will be described how to run the Examiner, an critic program for planning of waterflows through powerstations and their reservoirs, written by Olof Wickström as part of his Master Thesis in the summer and autumn of 1988 at the Department of Automatic Control, Lund Institute of Technology and Sydkraft AB.

If you are using a VAX 8530 under VMS version 4.7 and C-Prolog version 1.5.

You start the prolog interpretator with the command

```
prolog -g 1000 -l 1900 -t 500
```

If you do not use these options the program will run out of memory.

The machine will answer with the message

```
C-prolog version 1.5

yes
| ?-
```

The last is C-prolog's command prompt.

Now you load your program by writing

```
consult('examiner.pl').
```

Do not forget to always put a full stop at the end of a command.

The interpretator's answer would look something like

```
examiner.pl consulted 7644 bytes in 2.28333 sec

yes
```

You start the program by writing

```
examine_week(jfn)
```

The answer would be

```
Inputfile:
```

Now you write the name of the file that contains the input data on the form

```
' VMS_filename ' .
```

A typical file looks like

```
100.
32.
[0,0,0,0,  0,0,0,0,0].
[0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,
65,65,65,65,65,  65,65,65,65,65,  65,65,65,65,  65,65,65,65,
65,65,65,65,65,  0,65,65,65,65,  65,65,65,65,17,
0,0,0,0,  0,0,0,0,0,0,  0,0,0,0,0,  65,65,65,65,65,
65,65,65,65,65,  65,65,0,0,  0,0,0,0,0,  0,0,0,0,0,
65,65,65,65,65,  65,65,65,65,65,  65,65,0,0,  0,0,0,0,0,
0,0,0,0,0,  65,65,65,65,65,  65,65,65,65,65,  65,65,0,0,
0,0,0,0,0,  0,0,0,0,0,0,  65,65,65,0,65,  65,65,65,0,65,
65,65,65,65,  0,0,0,0,0,  0,0,0,0,0 ].
see(user),
seen.
```

Where

```
100
```

is the number of the first day of the week that should be criticized, with 1 as January the first,

```
32
```

is the average waterflow,

```
[ 0,0,0,0,  0,0,0,0,0]
```

is the flow the nine hours before the week that should be criticized starts,

and

```
[0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,
65,65,65,65,65,  65,65,65,65,65,  65,65,65,65,
65,65,65,65,65,  65,65,65,65,65,
0,65,65,65,65,  65,65,65,65,17,  0,0,0,0,  0,0,0,0,0,
0,0,0,0,0,0,  65,65,65,65,65,  65,65,65,65,65,  65,65,0,0,
0,0,0,0,0,  0,0,0,0,0,0,  65,65,65,65,65,  65,65,65,65,65,
65,65,0,0,  0,0,0,0,0,0,  0,0,0,0,0,0,  65,65,65,65,65,
65,65,0,0,  0,0,0,0,0,0,  0,0,0,0,0,
65,65,65,0,65,  65,65,65,0,65,  65,65,65,65,  0,0,0,0,0,
0,0,0,0,0 ].
```

is the flows for each hour of the week that should be criticized.

Observe that all . [ ] , are necessary and that the two lines

```
see(user).
seen.
```

must be at the end of the file.

The program will now print out the day, the hour, the flow, and the reservoir contents with possible error message for each hour in the week.

If you want to leave the interpretator after that the program has terminated you should write

```
(halt).
```