



LUND UNIVERSITY

A Multiprocessor DDC-Package

Breidegard, Björn; Hägglund, Tore; Gutman, Per-Olof; Nyqvist, Jean

1981

Document Version:

Publisher's PDF, also known as Version of record

[Link to publication](#)

Citation for published version (APA):

Breidegard, B., Hägglund, T., Gutman, P.-O., & Nyqvist, J. (1981). *A Multiprocessor DDC-Package*. (Technical Reports TFRT-7216). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:

4

General rights

Unless other specific re-use rights are stated the following general rights apply:

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: <https://creativecommons.org/licenses/>

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

A MULTIPROCESSOR DDC-PACKAGE

BJÖRN BREIDEGARD

TORE HÄGGLUND

PER-OLOF GUTMAN

JEAN NYQVIST

DEPARTMENT OF AUTOMATIC CONTROL
LUND INSTITUTE OF TECHNOLOGY
APRIL 1981

LUND INSTITUTE OF TECHNOLOGY DEPARTMENT OF AUTOMATIC CONTROL Box 725 S 220 07 Lund 7 Sweden		Document name	
		Internal report	
		Date of issue	
		April 1981	
Author(s) Björn Breidegard Tore Hägglund Per-Olof Gutman Jean Nyqvist		Document number	
		LUTFD2/(TFRT-7216)/0-065/(1980)	
		Supervisor	
		Sponsoring organization	
Title and subtitle			
A multiprocessor DDC-package			
Abstract			
<p>A one-processor DDC-package is modified to a multiprocessor DDC-package. It can be used e.g. in a hierarchical computer configuration, but other configurations are also possible. The programming language is Concurrent Pascal.</p>			
Key words			
Classification system and/or index terms (if any)			
Supplementary bibliographical information			
ISSN and key title			ISBN
Language	Number of pages	Recipient's notes	
English	65		
Security classification			

DOKUMENTDATABLAD RT 3/81

Distribution: The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis Lund.

A MULTIPROCESSOR DDC-PACKAGE

Björn Breidegard
Tore Hagglund
Per-Olof Gutman
Jean Nyqvist

Department of Automatic Control
Lund Institute of Technology
May 27, 1980

CONTENTS

CONTENTS

1. Introduction
 2. The monitor diagram
 3. DDCPAA Documentation
 4. OPCOM Documentation
 5. REGULATOR Documentation
 6. Transmission of text between two computers
 7. Transmission of data between two computers
 8. The new command SYNC
 9. Suggestions for improvements
 10. How to start up
- Appendix 1: DDCPAA
2: OPCOM
3: REGULATOR

INTRODUCTION

INTRODUCTION

Our work is based on the existing one-processor package DDCPAC, OPCOM, REGUL (see e.g. Tommy Essebo et al.: Facilities for concurrent processes in Pascal) which is assumed to be known by the reader.

We decided that any configuration of computers should be allowed, under the following conditions:

1. Each computer is started up by a terminal which is removed after the start-up (this restriction is caused by the limited number of ports).
2. One computer will retain the terminal during the running of the DDC-package. This computer must be called MAIN.
3. Any communication set-up between the computers is allowed, as long as it is consistent (i.e. if a message is received in the left port to be transmitted to another computer, it must not be sent on via the left port). Allowed configurations include:



See also the example of the chapter "How to start up".

The principle of our work was to change as little as possible in the existing DDC-package. As the original DDC-package was not constructed with intercomputer communication in mind, this means that there is a lot of room to improve our packaged. Some ideas are included in the chapter "Suggestions for improvements". These are left with a warm hand to our successors.

Our contribution mainly consists of

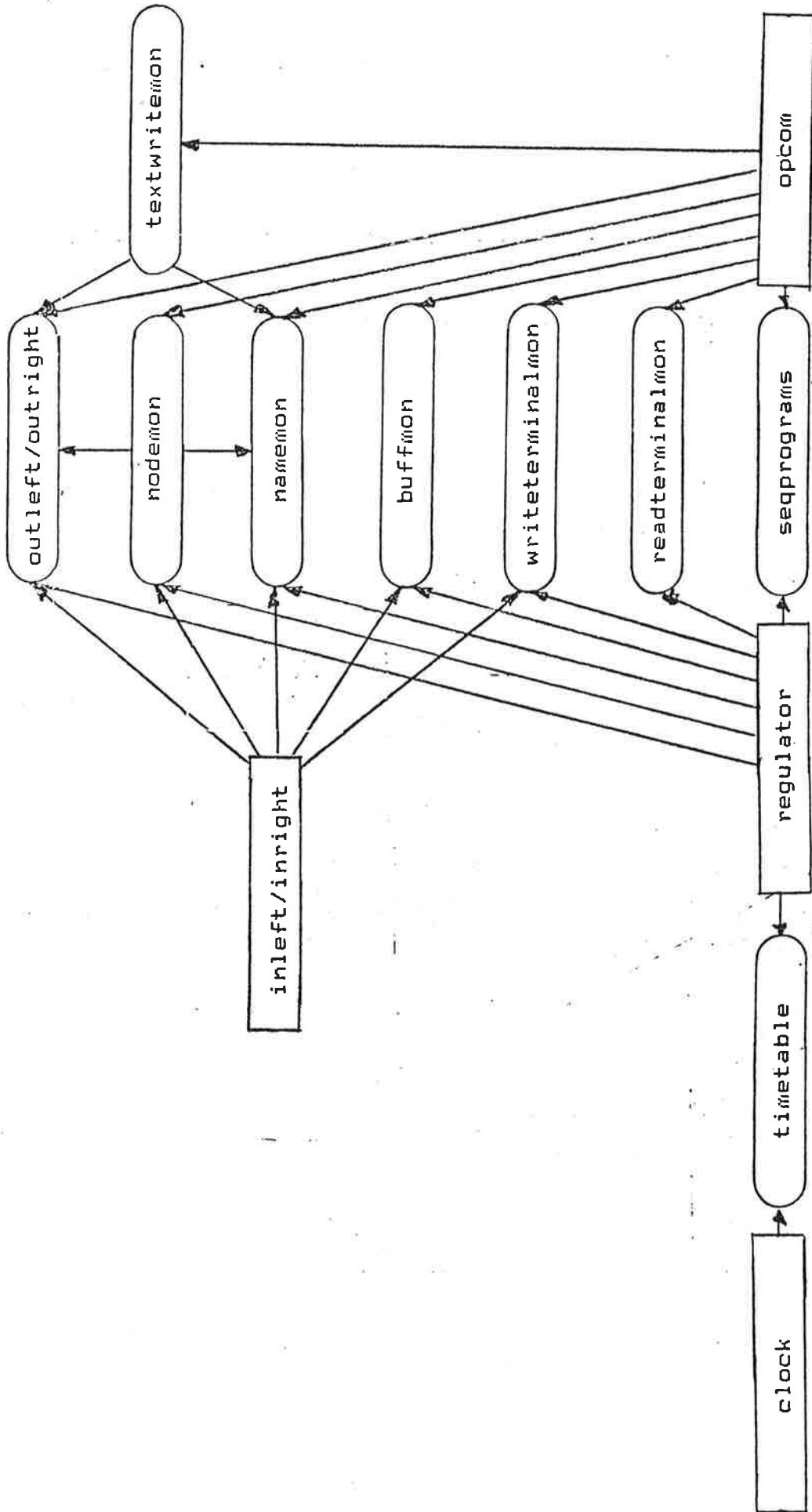
- a. The code performing the transmission of data and text between the computers (see ch. 6 and 7, and the monitors `namemonitortype`, `buffmontype`, `outleftmontype`, `outrightmontype`, `inlefttype`, `inrighttype`).
- b. Code for initialization. The initialization was messy because the ports have different names if they lead to the terminal or to another computer (see point 1 above). See also the monitors `namemonitortype`, `outleftmontype`, `outrightmontype`, `inlefttype`, `inrighttype`, and procedure `initialize` of OPCOM.

INTRODUCTION

- c. Necessary changes in the ddonodetype.
- d. An ad hoc facility for synchronization of nodes in different computers (see ch. 8).

We are indebted to Leif Andersson and Sven Erik Mattson who wrote the kernels that enabled our program, and who advised and aided us in the best possible way.

MONITOR DIAGRAM



DDCPAA DOCUMENTATION

```
{*****  
* namemontotype *  
*****}  
  
>Set of procedures which set a computer's name  
>or return information on a computer's name.  
  
procedure entry setmyname(var thisname:nametype);  
>This procedure, called by OPCOM, sets myname to  
>the name of this computer.  
  
procedure entry getmyname(var thisname:nametype);  
>This procedure returns the name of this computer.  
  
procedure entry getmyname1(var thisname:nametype);  
>Same as getmyname. Called by the READ procedure  
>in OPCOM.  
  
procedure entry setleftnames(var thisname:nametype;  
var i:integer);  
>This procedure, called by OPCOM, assigns names to  
>the left computers.  
  
procedure entry setrightnames(var thisname:nametype;  
var i:integer);  
>This procedure, called by OPCOM, assigns names to  
>the right computers.  
  
procedure entry checkcomp(var thisname:nametype;  
var answer:answertype);  
>This procedure informs the caller if thisname is  
>the name of this computer, if it is the name of  
>a left or a right computer, or if there is no  
>computer by that name.  
  
{*****  
* buffmontype *  
*****}  
  
>This monitor consists of a ring buffer for text to  
>be used by the OPCOM of computers other than MAIN.  
  
procedure entry write(var ch:char);  
>This procedure, called by INLEFT and INRIGHT, puts  
>a character into the ring buffer.  
  
procedure entry read(var ch:char);  
>This procedure, called by OPCOM, fetches a  
>character from the ring buffer.  
  
{*****  
* outleftmontype *  
*****}  
  
>This monitor sends text and data on the left channel.
```

DDCPAA DOCUMENTATION

```
procedure entry setxleft(var dev:iodevice);
>This procedure, called by OPCOM, designates one
>of the output channels as the left channel.

procedure entry getxleft(var dev:iodevice);
>This procedure, called by INLEFT, returns the
>channel number of the left channel.

procedure entry dataout(var comp,node:nametype;var outval:univ rstring);
>This procedure sends data on the left channel.

procedure entry asciiout(var comp:nametype;var ch:char);
>This procedure sends text on the left channel.

{*****}
* outrightmontype *
*****}

>This monitor is the same as OUTLEFTMONTYPE
>with right substituted for left.

{*****}
* readterminalmontype *
*****}

procedure entry read(var character:char);
>This procedure reads from the keyboard when myname=MAIN.
>It is not used in the other computers.

{*****}
* writeterminalmontype *
*****}

procedure entry write(var ch:char);
>This procedure writes on the terminal when myname=MAIN.
>It is not used in the other computers.

{*****}
* textwritemontype *
*****}

procedure entry textwrite(var dest:nametype;var ch:char);
>This procedure, called by OPCOM if myname=MAIN,
>sends text generated on the terminal to computers
>other than MAIN.

begin
  namemon.checkcomp(dest;answer);
  case answer of
    left: outleft.asciiout(dest;ch);
    right: outright.asciiout(dest;ch)
  end;
end;
```

DDCPAA DOCUMENTATION

```
{*****}
* seqprogramstype *
*****}

>This monitor is used at system startup only.

{*****}
* nodemonitortype *
*****}

ddcnodetype=record
>See definition in program REGULATOR.

>This monitor is a set of procedures which handles
>operations performed on the nodes.

procedure entry regputgetnode(var node:ddcnodetype;
                               var anyfound:boolean);
>This procedure, called by REGULATOR, transfers
>nodes between the active list and the work space
>of REGULATOR.

procedure entry getoutvalue(var number:integer;
                             var outval:real);
>This procedure, called by REGULATOR, returns the
>output value from the node with the given pointer.

procedure lookup(var name:nametype; var ptr:integer);
>This procedure returns the pointer to the node with
>the given name.

procedure entry getrealtime(var rtime:integer);

procedure entry putoutvalue(var node:nametype;var value:univ real);
>This procedure, called by INLEFT and INRIGHT, sets
>the outvalue of an innode.

procedure entry opgetnode(var node:ddcnodetype;
                          var notfound,notspace:boolean);
>This procedure, called by OPCOM in response to
>the command OPEN, copies an existing node into
>the work space of OPCOM or informs the caller
>that no node exists by that name.

procedure entry delete(var nodename:nametype;var result:integer);
>This procedure, called by OPCOM in response to
>the command DELETE, transfers a node from the
>active to the inactive list.

procedure entry link(var node:ddcnodetype);
>This procedure, called by OPCOM in response to
>the command LINK, copies a node from the work
>space of OPCOM to a node in the active list.
```

DDCPAA DOCUMENTATION

```

procedure entry checkname(var name:name;
    var ptr,ptri:integer);
>This procedure returns the pointer to and the
>priority of the node with the given name.

procedure entry datawrite(var comp,node:name;var outvalue:real);
>This procedure, called by REGULATOR, directs data
>to a node in the appropriate computer.

{*****
 * timetable *
*****}

>This monitor synchronizes the action of the
>REGULATOR with the realtime clock.

procedure entry wait;
>This procedure, called by REGULATOR, delays regqueue.

procedure entry schedule;
>This procedure, called by CLOCK, continues regqueue.

{*****
 * inlefttype *
*****}

>This process reads continuously on the left
>channel and sends the information to the
>appropriate place.

```

DESTINATION	ASCII	NUMERIC
not myself	outright	
myself(=MAIN)	writeterminalmon.write	
myself(<>MAIN)	buffmon.write	
		putoutvalue

```

inlefttype=process(node:mon;node:mon;name:mon;name:mon;name:mon;name:mon;name:mon;
    buffmon:buffmon;outleft:right;outright:right;
    outleft:right;outright:right;
    writeterminalmon:writeterminalmon);

{*****
 * inrighttype *
*****}

>This process is the same as INLEFTTYPE with
>right substituted for left.

```

DDCPAA DOCUMENTATION

```
{*****  
* clocktype *  
*****}
```

>This process is the alarmclock for TIMETABLE.

```
type clocktype=process(timetable:timetabletype);
```

```
{*****  
* regulatortype *  
*****}
```

```
regulatortype=process(buffmon:buffmontype;  
  readterminalmon:readterminalmontype;  
  writeterminalmon:writeterminalmontype;  
  seqprograms:seqprogramstype;  
  nodemon:nodemonitortype;  
  timetable:timetabletype;  
  namemon:namemonitortype;  
  outleft:outleftmontype;  
  outright:outrightmontype);
```

```
procedure entry read(var character:char);  
  if myname=MAIN then readterminalmon.read(character)  
  else buffmon.read(character)
```

```
procedure entry write(var character:char);
```

```
  case MAIN=  
    myself: writeterminalmon.write(character);  
    left: outleft.asciout(MAIN,character);  
    right: outright.asciout(MAIN,character);
```

```
  procedure entry wait;  
    timetable.wait
```

```
  procedure entry regputgetnode(var node:ddcnodetype;  
    var anyfound:boolean);  
    nodemon.regputgetnode(node,anyfound)
```

```
  procedure entry getoutvalue(var number:integer;  
    var outvalue:real);  
    nodemon.getoutvalue(number,outvalue)
```

```
  procedure entry getrealtime(var rtime:integer);  
    nodemon.getrealtime(rtime)
```

```
  procedure entry datawrite(var comp,nodename:nametype;  
    var sendvalue:real);
```

```
  case comp is to the  
    left: outleft.dataout(comp,nodename,sendvalue);  
    right: outright.dataout(comp,nodename,sendvalue);
```

```
  procedure entry getmyname(var thisname:nametype);  
    namemon.getmyname(thisname)
```

DDCPAA DOCUMENTATION

```
{*****  
* opcomtype *  
*****}  
  
opcomtype=process(buffmon:buffmontype;  
  readterminalmon:readterminalmontype;  
  writeterminalmon:writeterminalmontype;  
  textwritemon:textwritemontype;  
  seqprograms:seqprognamstypel;  
  nodemon:nodemonitontype;  
  namemon:namemonitontype;  
  outleft:outleftmontype;  
  outright:outrightmontype);  
  
procedure entry read(var character:char);  
  if myname=MAIN then readterminalmon.read(character)  
  else buffmon.read(character);  
  
procedure entry write(var character:char);  
  case MAIN=  
  myself: writeterminalmon.write(character);  
  left:  outleft.asciout(MAIN,character);  
  right: outright.asciout(MAIN,character)  
  
procedure entry opgetnode(var node:ddcnodetype;  
  var notfound,notspace:boolean);  
  nodemon.opgetnode(node,notfound,notspace)  
  
procedure entry link(var node:ddcnodetype);  
  nodemon.link(node)  
  
procedure entry delete(var nodename:nametype;  
  var result:integer);  
  nodemon.delete(nodename,result)  
  
procedure entry checkname(var name:nametype;  
  var ptr,ptri:integer);  
  nodemon.checkname(name,ptr,ptri)  
  
procedure entry textwrite(var dest:nametype;  
  var ch:char);  
  textwritemon.textwrite(dest,ch);  
  
procedure entry checkcomp(var thisname:nametype;  
  var answer:answertype);  
  namemon.checkcomp(thisname,answer);  
  
procedure entry setmyname(var myname:nametype);  
  namemon.setmyname(myname);  
  
procedure entry setleftnames(var thisname:nametype;  
  var i:integer);  
  namemon.setleftnames(thisname,i);
```

DDCPAA DOCUMENTATION

```
procedure entry setrightnames(var thisname:nametype;  
    var i:integer);  
    namemon.setrightnames(thisname,i);
```

```
procedure entry setxleft(var dev:iodevice);  
    outfile.setxleft(dev);
```

```
procedure entry setxright(var dev:iodevice);  
    outfile.setxright(dev);
```

```
{*****  
 * main *  
 *****}
```

```
var readterminalmon:readterminalmontype;  
    writeterminalmon:writeterminalmontype;  
    textwritemon:textwritemontype;  
    seqprograms:seqprogramstypel;  
    nodemon:nodeмонтortype;  
    timetable:timetabletype;  
    clock:clocktype;  
    regulator:regulatortype;  
    opcom:opcomtype;  
    namemon:namemontortype;  
    buffmon:buffmontype;  
    outfile:outleftmontype;  
    outfile:outrightmontype;  
    infile:inlefttype;  
    infile:inrighttype;
```

OPCOM DOCUMENTATION

program opcom;
entryprocedures=opgetnode,delete,link,checkname,setmyname,
setleftnames,setrightnames,checkcomp,
textwrite,setxleft,setxright
>See comments in main program - DDCPAA.

ddcnodetype=record
>See definition in program REGULATOR.
procedure error(err:errors);
>This procedure writes error messages on the terminal.

procedure writeaddress(addr:address);
>This procedure, called by SHOW, writes an address on the
terminal.

procedure readname;
>This procedure reads a node name from the terminal.

procedure open;
>This procedure copies a node into the work space of
>OPCOM if a node by that name exists, otherwise
>a new node is created.

procedure show;
>This procedure displays on the terminal the values of
>the node in the work space of OPCOM.

procedure link;
>This procedure copies the node in the work
>space of OPCOM into the active list.

procedure delete;
>This procedure deletes a node from the active list.

procedure readreal(var variable:real;
tag:nodetype);
>This procedure reads a real number from the keyboard
>if the program is running in MAIN. If the program is
>running in a program other than MAIN, a real number
>is read from the ring buffer.

procedure readint(var variable:integer);
>This procedure reads an integer. See READREAL.

procedure readaddr(var signal:address;
tag:nodetype;
outsignal:boolean);
>This procedure reads an address. See READREAL.

procedure readcomp(var comp:nametype);
>This procedure reads a computer name. See READREAL.

procedure setvariable;
>This procedure changes values in the parampart of a node.

OPCOM DOCUMENTATION

```

procedure initialize;
>This procedure initializes the variables of OPCOM
>and asks for the structure of the communication link
>relative to this computer (which computers lie to
>the left, which computers lie to the right, and what
>is the name of this computer).
begin {opcom}
  initialize;
  if myname=MAIN then
    repeat
      i:=0;
      while not eoln do begin
        read(ch);
        i:=i+1;
        command[i]:=ch;
        buff[i]:=ch;
      end;{while}
      if command='BYE' then
        repeat
          write('>');
          read(actcomp);
          checkcomp(actcomp,exist);
          if exist=nonexistent then writeln('no computer name')
          else write(actcomp,>')
          until exist<>nonexistent
        else if actcomp=MAIN then
          begin
            case command of
              open : open;
              show : show;
              link : link;
              delete : delete;
              other : setvariable
            end;{case}
            write(myname,>')
          end
        else begin
          while not eoln do begin
            i:=i+1;
            read(buff[i]);
          end;{while}
          for j:=1 to i do textwrite(actcomp,buff[j]); >Send the complete
            command to the
            appropriate computer.
          readln;
          until false
        else repeat
          read(command);
          case command of
            open : open;
            show : show;
            link : link;

```

OPCOM DOCUMENTATION

```
delete : delete;  
other : setvariable  
end:{case}  
write(myname,'>')  
until false;  
end.{opcom}
```

REGULATOR DOCUMENTATION

```

program regulator;
function adin(chan:integer):real; external;
>Analog-to-digital input.

procedure daout(chan:integer; value:real); external;
>Digital-to-analog output.

entryprocedures=wait,regputgetnode,getoutvalue,
getrealtime,datawrite,getmyname
>See comments in main program - DDCPAA.

```

DEFINITION OF NODE STRUCTURES

```

type nametype=array[1..10] of char;

address=record
  name:nametype;
  number:integer
end;{address}

nodetype=(innode, PIDnode, outnode);

paramtype=record
  goalcomp:nametype;
  intime,outtime,lasttime:integer;
  case tag:nodetype of
    innode:(insig,output,address;
             scale,filter:real);
    PIDnode:(PIDin,PIDref,PIDout,address;
             kti,td,alpha,beta,limit:real);
    outnode:(insig1,insig2,insig3,
             insig4,output,address;
             scal1,scal2,scal3,scal4,level:real)
  end;{paramtype}

statetype=record
  filter,stateldd,ipart,dpert:real;
end;{statetype}

ddcnodetype=record
  fwd,bwd:integer;
  name:nametype;
  priority:integer;
  period,counter:integer;
  outvalue:real;
  parampart:paramtype;
  state:statetype
end;{ddcnodetype}

function ulim(u,lo,lim,hi:real):real;
>This function puts an upper and lower limit
>on the signal u.

```

REGULATOR DOCUMENTATION

function nodevalue(var addr:address):real;
>This function gets a value from the AD-converter
>or from the outvalue of the addressed node.

procedure setouttime(var intime,outtime,lasttime,
period:integer);

>This procedure, called by INREG, puts the realtime
>at which the outvalue is taken from a node into
>outtime. (For more information, see the chapter
>on synchronization.)

procedure innreg;

>This procedure regulates innodes.

procedure PIDreg;

>This procedure regulates PID-nodes.

procedure outreg;

>This procedure regulates outnodes.

begin {regul}
getmyname(myname);

repeat

wait;

regputgetnode(node,anyfound);

>Get the name of this computer.

>Wait until the next clock tick.

>Move the node in the work space of
REGULATOR back to the active list, and
move the next node with counter<=0 to
the work space of REGULATOR.

while anyfound do begin

case nodetype of

innode: innreg;

PIDnode: PIDreg;

outnode: outreg

end{case}

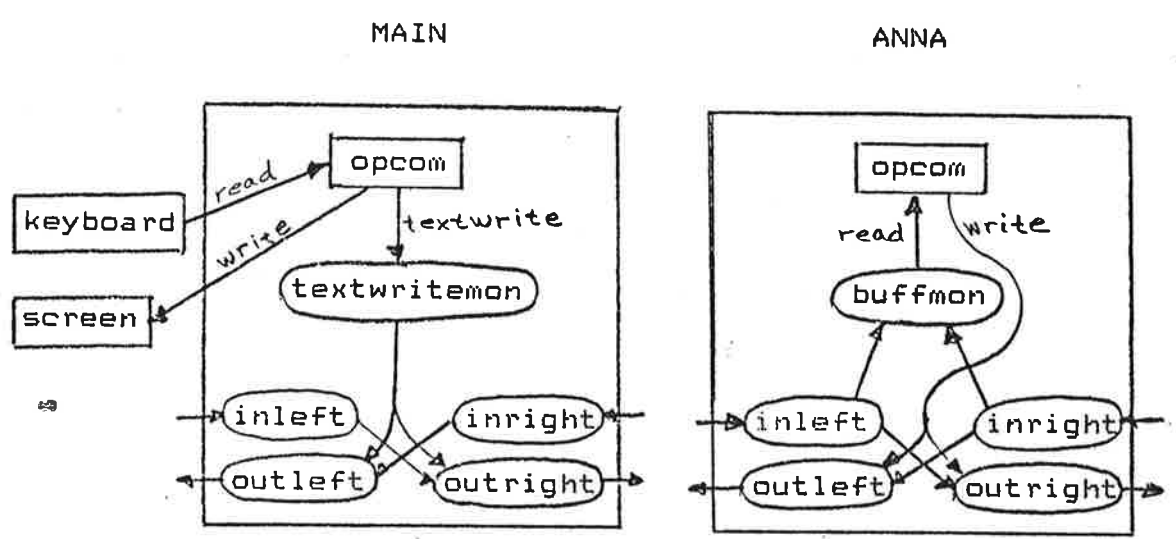
regputgetnode(node,anyfound)

end {while}

until false;

end {regul}.

Transmission of text between two computers.



Suppose that we want to open the node PID in the computer ANNA.

ANNA >OPEN PID

The string OPEN PID is sent from OPCOM in MAIN character by character. E.g. the O is sent by the command

```
textwritemon.textwrite('ANNA      ','O')
```

In textwritemon, the procedure checkcomp in namemon is called to check whether ANNA is to the left or to the right.

```
namemon.checkcomp('ANNA      ',answer)
```

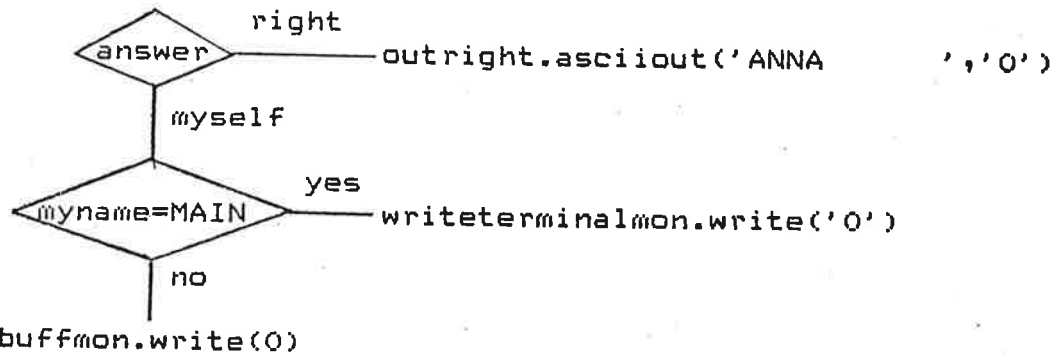
asciout is then called in outleft or outright depending on if the answer is left or right.

```
outright.asciout('ANNA      ','O')
```

In outright, the following 12 characters are sent on the right channel by io-routine: @ (to inform the receiver that text and not data is coming), A, N, N, A, , , , , , , , O.

In ANNA the ^{process} ~~monitor~~ inleft takes care of the information. The command checkcomp in namemon is called to check if the message is meant for ANNA or another computer.

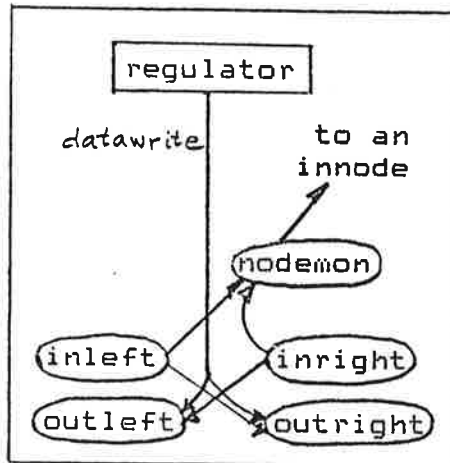
```
namemon.checkcomp('ANNA      '.answer)
```



In this case the character O is put into the ringbuffer in buffmon.

Finally the readcommand, that continuously checks if there is something to fetch from buffmon, brings the O into OPCOM. At the same time, an O is sent back to the screen from ANNA, to make it possible for the operator to check that the message is correctly transferred.

Transmission of data between two computers.



Suppose that we want to send the value 7.35 from the outnode OUT in ANNA to the innode IN in BRITTA. (We are not allowed to send to anything but innodes in other computers).

In the parampart of OUT, GOALCOMP must be set to BRITTA and OUTPUT to IN. In IN, INSIG must be set to EXT.

The transmission is initialized by the command

```
datawrite('BRITTA', 'IN', 7.35)
```

in ANNA. The command

```
namemon.checkcomp('BRITTA', answer)
```

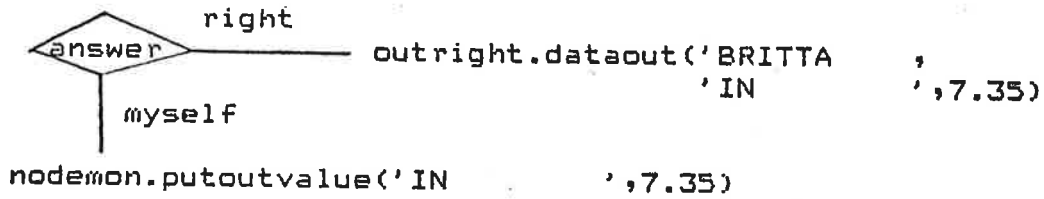
checks whether BRITTA is to the left or to the right. If the answer is right, the procedure

```
outright.dataout('BRITTA', 'IN', 7.35)
```

will be called. In outright, the last variabel is declared as a universal string, and not as real. The real value is represented by 4 bytes, and they will be sent as characters. The following 25 characters are now sent on the right channel by the io-routine: # (to inform the receiver that data and not text is comming), B, R, I, T, T, A, , , , , I, N, , , , , , , , value[1], value[2], value[3], value[4].

In BRITTA, inleft takes care of the incoming information. The command checkcomp in namemon is called to check if the information is meant for BRITTA.

```
namemon.checkcomp('BRITTA',answer)
```



Finally putoutvalue in nodemon sets outvalue in IN equal to 7.35.

THE NEW COMMAND SYNC

Background

Every node produces, as a result of the computations when it is implemented, a real number: the outvalue (see the definition of ddcnodetype). The outvalue has one or more destinations - another node in the same computer, the DA-converter of the same computer, or a node in another computer. Moreover the outvalue is always put into the field "outvalue" of the ddcnodetype record.

Also, the AD-converter produces real numbers destined for node(s) in the same computer.

Modes of transport

There are two ways of transporting the outvalue (AD-value) to its destination, an active mode, and a passive mode.

Definition: The active mode of transport (A) means that the producing node actively sends the outvalue to the destination.

Definition: The passive mode of transport (P) means that the producing node (AD-converter) only sets the field outvalue (corresponding) and that the destination has to pick up the outvalue there.

The different ways of transport are used as follows (note that some routes are prohibited: x):

<u>Destination</u>		<u>Source:</u>		
The same computer	Another computer	IN-node	PID-node OUT-node	AD-conv
IN-node PID-node OUT-node			P	P
DA-conv			A	x
	IN-node		A	x
	PID-node OUT-node DA-conv		x	x

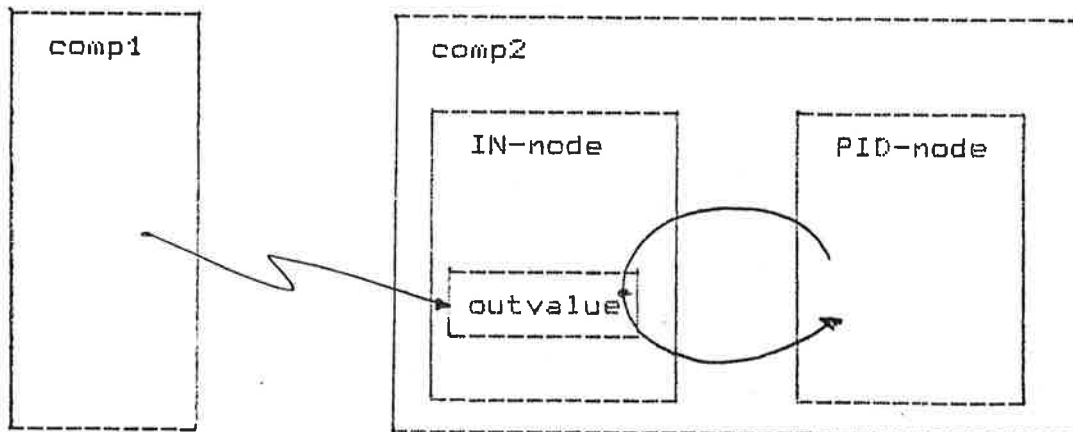
SYNC

When a value is actively sent to an IN-node of another computer, this IN-node solely serves as a station of reception. The IN-node gets the value into the field outvalue, and does not process it (it can, however, send it on to a DA-converter, or to an IN-node of yet another computer). See the procedure inreg of REGULATOR.

The problem

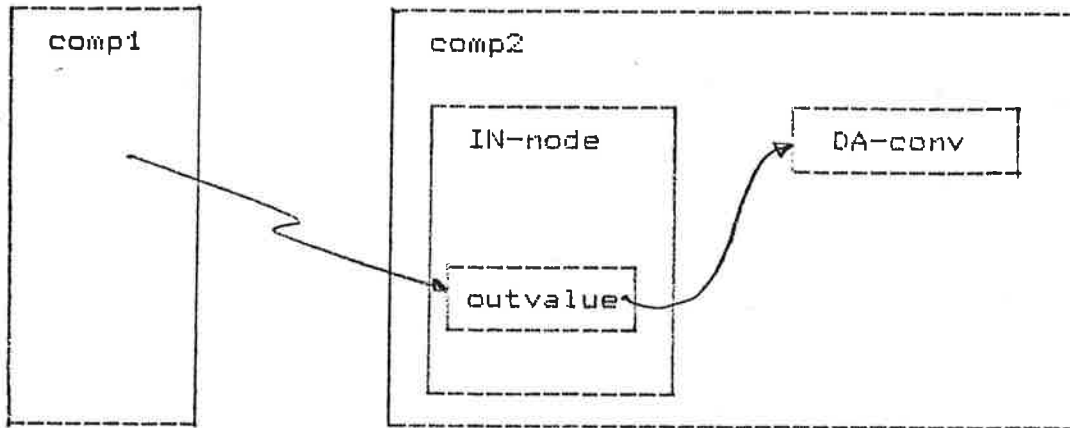
Typically the following synchronization problem occurs:

Case_1: A node in comp1 sends actively a value to an IN-node of comp2. The value now rests in the field outvalue waiting for e.g. a PID-node of comp2 to pick it up. The delay between the time instant when the value was placed in the field outvalue of the IN-node and the time instant it is picked up by the PID-node may cause severe problems. Clearly a synchronization between the time of reception by the IN-node (intime) and the time when the value is picked up (outtime) is needed.



Case_2: The same problem occurs if the IN-node in comp2 is to send its outvalue to a DA-converter. The interval between the time of reception (intime) and the time when the IN-node is interpreted, and thus sends the value to the DA-converter (outtime) should be as short as possible.

SYNC



It is not a trivial task to synchronize the clocks of the computers. If this was done, it would be possible to ensure that the counters (see the record `ddcnodetype`) of the connected nodes in `comp1` and `comp2` were synchronized with a predetermined lag, so that the nodes are executed in the correct order with minimum delay. If the clocks are not synchronized it seems to be quite difficult to synchronize the counters.

Another problem is how e.g. a temporary traffic jam on the link connecting `comp1` and `comp2` which causes a delay in the transfer of the value to the `IN-node` of `comp2`, should influence the execution of nodes in `comp2` needing the value. Should the execution be delayed, and, if so, for how long? This problem has not been solved, but a field `lasttime` is included in the `ddcnodetype`. This field contains the last time instant the regular data transfer went wrong (see below).

Principle of solution

It was decided not to try to synchronize the different computers. As the aim is to minimize the delay between `intime` and `outtime`, all actions could be taken in `comp2`. The following situations may occur both in case 1 and case 2:

```
intime  *-----*-----*----- /ticks/
outtime -*-----*-----*----- /ticks/
```

Acceptably small delay
(time between stars = sampling period)

```
intime  *-----*-----*----- /ticks/
outtime -----*-----*-----*----- /ticks/
```

Unacceptably long delay

SYNC

Let the operator have access to the field `intime` and `outtime` of the receiving IN-node when commanding `OPEN-SHOW`. He will then get the last `intime` and the last `outtime`, at the occasion the command `OPEN` was executed. He will get one of the following four possible pictures:

a)	<code>intime</code>	<code>*-</code>	b)	<code>intime</code>	<code>-----*</code>
	<code>outtime</code>	<code>-*</code>		<code>outtime</code>	<code>*-----</code>
c)	<code>intime</code>	<code>*-----</code>	d)	<code>intime</code>	<code>----*</code>
	<code>outtime</code>	<code>-----*</code>		<code>outtime</code>	<code>*---</code>

Knowing the sample period, the operator has no trouble to distinguish which of the cases a, b, c, d belong to the case of acceptably small delay respectively the case of unacceptably long delay.

By changing the counter of the node in `comp2` that transfers the value (case 1: the PID-node, case 2: the IN-node itself), the time instant of the execution of the node is changed, and the delay between `intime` and `outtime` is altered. This is done by the commands `OPEN-SYNC-LINK`, see below.

The problem of occasional irregularities in `intime` has been tackled in the following way. Assume that you have been able to synchronize the `intime` and `outtime` nicely. Sometimes, however, the transfer of data gets delayed:

```
intime  *-----*-----*-----*-----*-----
outtime -*-----*-----*-----*-----*-----
                ↑
```

In IN-node of `comp2` the field `lasttime := outtime`, if $(outtime - intime) \geq period/2$. See the procedure `setouttime` in `REGULATOR`, and procedure entry `getoutvalue` of `nodemonitortype` in `DDCPAA`. In the above figure `lasttime` will be set at the arrow.

By commanding `OPEN-SHOW` for the IN-node of `comp2` a few times, the operator can get an idea if and how often `lasttime` is updated. If it occurs too often, he can adjust `outtime` by the command `SYNC`, as mentioned above.

Implementation

`Ddcnodetype` has been changed to include `intime`, `outtime`, and `lasttime`.

`Intime` is set in procedure entry `putoutvalue` of `nodemonitortype` in `DDCPAA`.

SYNC

Outtime and lasttime are set by either procedure entry getoutvalue of nodemonitortype in DDCPAA (case 1) or by the procedure setouttime of REGULATOR (case 2).

Intime, outtime, lasttime take on integer values denoting real time in number of ticks.

The procedure show of OPCOM has been changed for the case of innode, to include a display of intime, outtime, and lasttime.

In procedure open of OPCOM the statement 250: node.counter:=0 has been included. This is because the field node.counter is used to store the argument of the command SYNC.

- SYNC <integer> is a new command in form of a "subcommand" that can be performed only after and OPEN-command has been executed. It is included in the variable vars of OPCOM. SYNC <integer> causes node.counter:=<integer>. See statement 430 in the procedure setvariable of OPCOM. See also the next paragraph.

Procedure entry link of DDCPAA has got a new statement 516: counter:=counter + node.counter. Thus, if SYNC was used before commanding LINK, the value of the counter of the node in the active list is changed, and the coming instants of execution of the node will be hastened or delayed. As is well known, each time the active list is gone through, counter is decreased by one. The node is executed when counter = 0. At this occasion counter := period. Therefore:

<integer> < 0 [> 0] hastens [delays] the next time instant of execution by <integer> ticks.

Command sequence

Case 1 and case 2 refer to the cases of the section "The problem". Underlined strings are generated by the program. String that are not underlined are generated by the operator.

```
>comp2
comp2>OPEN IN-node
comp2>SHOW
```

```
PERIOD_____10
INTIME_____2370
OUTTIME_____2374
LASTTIME_____34
```

SYNC

A delay of 4 ticks is unacceptable. Now case 1 and case 2 differ.

Case_1

We must change the counter of the PID-node that fetches the value.

```
comp2>>OPEN PID-node
comp2>>SYNC -3
comp2>>LINK
```

Now we are ready to inspect the result:

```
comp2>>OPEN IN-node
comp2>>SHOW
```

```
INTIME-----3010
OUTTIME-----3011
LASTTIME-----34
```

OK!

Case_2

The IN-node itself sends the value to a DA-converter (or another computer). We must change the counter of the IN-node.

```
comp2>>SYNC -3
comp2>>LINK
```

We are ready to inspect the result. Do not forget to OPEN again in order to get a fresh version of the IN-node of the active list.

```
comp2>>OPEN IN-node
comp2>>SHOW
```

```
INTIME-----3010
OUTTIME-----3011
LASTTIME-----34
```

OK!

SUGGESTIONS FOR IMPROVEMENTS

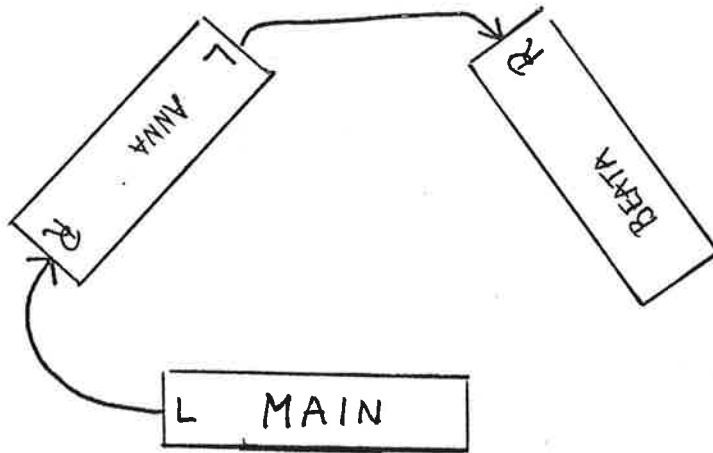
- 1) Every time a message is sent, the message must be preceded by the name of the destination computer. If computer names were shortened from ten letters to one letter, there would be less overhead time in sending messages.
- 2) There is a trade-off between sending one character of text or data at a time and sending a fixed string length every time. At present, only one character of text or data is sent at a time. An implementation which would send a variable string length would be preferable.
- 3) There are several places in the program where error routines have not been implemented.
- 4) There is no directory of node names. Once nodes have been created, their names must be remembered.
- 5) There is no way to check the structure of the communication link (where computers are relative to each other) after the structure has been initialized.
- 6) It would be desirable to start up the DDC programs in all the computers from the MAIN computer, rather than from a local terminal which is then disabled.
- 7) The hardware buffer used to send messages between computers might have to be enlarged.
- 8) There exists no synchronization facility between nodes in the same computer (cf. the PDP-15 version of the DDC-package).

HOW TO START UP

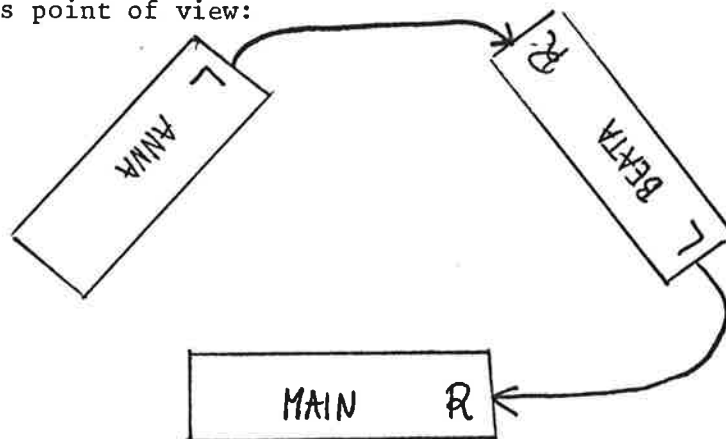
We assume that you want to create the following computer network:

The computer on which the terminal will be attached when you finally run the DDC-package must be called MAIN.

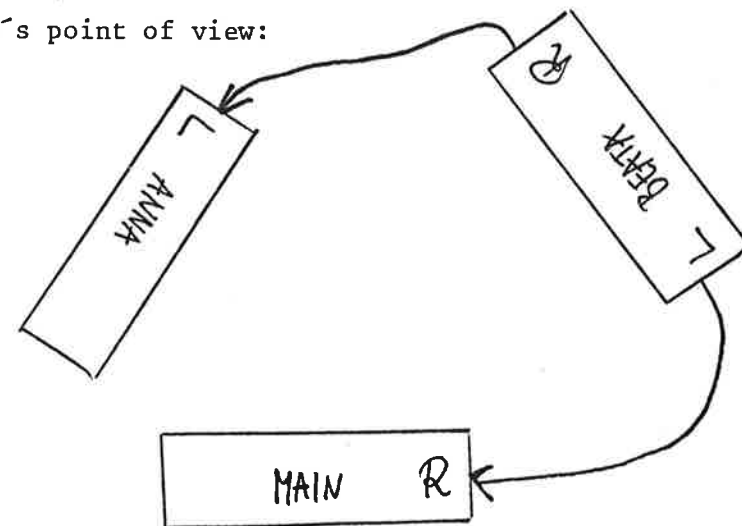
The network from MAIN's point of view:



From ANNA's point of view:



From BETA's point of view:



L and R stand for Left port and Right port, respectively. The arrows should be interpreted as follows: Messages from MAIN to both ANNA and BEATA will be shipped via the Left port of MAIN. Messages from BEATA to MAIN are sent through the Left port of BEATA, messages from BEATA to ANNA are sent through the Right port of BEATA.

When constructing this communication diagram, make sure that you don't create a "circular mess", i.e. if you want the messages from ANNA to MAIN to go via BEATA, you cannot let the messages from BEATA to MAIN go via ANNA.

The discette must contain the following files:

- KERNEL.SAV
- KERN3.SAV
- OPCOM3.REL
- REGUL3.REL
- DDCPAA.EMU

Start up of ANNA:

DC OFF, POWER ON, DC ON
 SET USR NOSWAP
 Load the discette
 Press ALPHA LOCK
 RUN DX1:KERNEL
 DX1:DDCPAA

SYSTEM READY

NAME OF LEFT COMPUTER 1: BEATA (if you want no left computer, press RETURN twice)
NAME OF LEFT COMPUTER 2: MAIN
NAME OF LEFT COMPUTER 3: Press RETURN
NAME OF RIGHT COMPUTER 1: Press RETURN
NAME OF THIS COMPUTER: ANNA

Remove the discette

Start up of BEATA:

Corresponding to the start up of ANNA.

Start up of MAIN:

See start up of ANNA for the first four points. Then
 RUN DX1:KERN3
 DX1:DDCPAA

SYSTEM READY

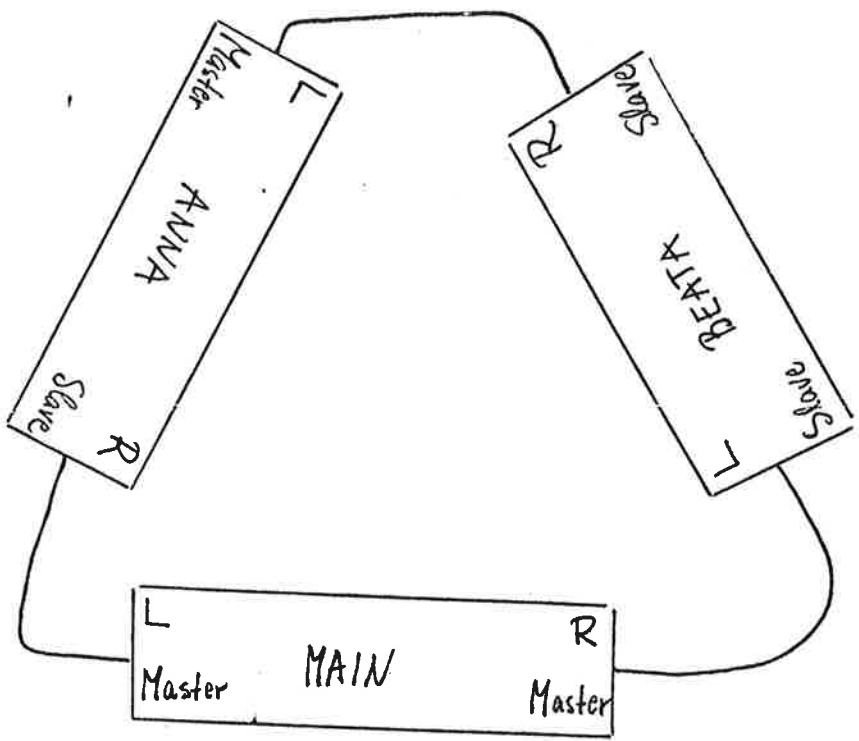
NAME OF LEFT COMPUTER 1: ANNA
 :
 :
NAME OF THIS COMPUTER: MAIN
MAIN >

(the underlined strings are the responses of the computer)

Now the computers are ready, but we must first connect them:

The terminal port of MAIN is connected to the terminal in the regular fashion.
 All other ports: Set the baud rate = 4800 (you might have to lower this figure later if the transmission load becomes too heavy). Set parity= none.

Every connector must have a MASTER port at one end and a SLAVE port at the other. The following set up is possible, for instance:



Command sequence:

```

MAIN> OPEN PID7           "Node manipulation commands in MIAN
MAIN> ...
:
:
MAIN> BYE                 "Leave MAIN
> BEATA                   "Choose BEATA
BEATA> ...                "Node manipulation in BEATA
BEATA> ...
:
:
BEATA> BYE                "Leave BEATA
>
...etc

```

Close down: DCOFF

REMEMBER: Keep track MANUALLY of the active list and the connections between the nodes.

```
1 const maxcomputers=4;
2 const namelength=10;
3 type tdevice=(s10,s11,s12);
4 type tparam=(input,output);
5 type nametype=array[1..10] of char;
6 type answerType=(right,left,myself,nonexistent);
7 type nodetype=(inode,PIDnode,outnode);
8 type rstring=array[1..4] of char;
9
10 {*****}
11 * namemonitorType*
12 {*****}
13
14 type namemonitorType = monitor;
15 type neighbours = array[1..maxcomputers] of nametype;
16 type superqueue = array[1..3] of queue;
17
18 var myname:nametype;
19 leftcomps:neighbours;
20 rightcomps:neighbours;
21 ready:boolean;
22 place:integer;
23 mynamequeue:superqueue;
24
25 procedure entry setmyname(var thisname:nametype);
26 begin
27   myname:=thisname;
28   ready:=true;
29   continue(mynamequeue[1]);
30 end;
31
32 procedure entry getmyname(var thisname:nametype);
33 begin
34   if not ready then begin
35     place:=place+1;
36     delay(mynamequeue[place]);
37   end;
38   thisname:=myname;
39   if not empty(mynamequeue[2]) then continue(mynamequeue[2]);
40   continue(mynamequeue[3]);
41 end;
42
43 procedure entry getmyname1(var thisname:nametype);
44 begin
45   thisname:=myname
46 end;
47
48 procedure entry setleftnames(var thisname:nametype;
49                               var i:integer);
50 begin
51   leftcomps[i]:=thisname
52 end;
```

```
53 procedure entry setrightnames(var thisname:nametype;  
54 var i:integer);  
55 begin  
56   rightcomps[i]:=thisname  
57 end;  
58  
59 procedure entry checkcomp(var thisname:nametype;  
60 var answer:answertype);  
61 var i:integer;  
62 done:boolean;  
63 begin  
64   answer:=noneexistent;  
65   done:=false;  
66   i:=1;  
67   if (thisname=myname) then answer:=myself  
68   else repeat  
69     if (thisname=leftcomps[i]) then begin  
70       answer:=left;  
71       done:=true  
72     end  
73   else if (thisname=rightcomps[i]) then begin  
74     answer:=right;  
75     done:=true;  
76   end;  
77   i:=i+1;  
78 until done or (i>maxcomputers)  
79 end; {checkcomp}  
80  
81  
82  
83 begin(namemonitorotype)  
84   myname:='MAIN';  
85   ready:=false;  
86   place:=0  
87 end; {namemonitorotype}  
88  
89 {*****  
90 * buffmontyps *  
91 *****}  
92  
93 type buffmontype=monitor;  
94 const bufflength=200;  
95 var xchar :array[1..bufflength] of char;  
96     inpos :integer;  
97     outpos :integer;  
98     buffqueue :queue;  
99 procedure entry write(var ch:char);  
100 begin  
101   inpos:=inpos+1;  
102   xchar[inpos]:=ch;  
103   if inpos=bufflength then inpos:=0;  
104   continue(buffqueue)  
105 end; {fwrite}
```

```
106 procedure entry read(var ch:char);
107 begin
108   if inpos=outpos then delay(buffqueue);
109   outpos:=outpos+1;
110   ch:=xchar[outpos];
111   if outpos=buflength then outpos:=0;
112   end;{read}
113   begin(buffmontype)
114     inpos:=0;
115     outpos:=0;
116   end;{buffmontype}
117
118   {*****}
119   * outleftmontype *
120   {*****}
121   {*****}
122   type outleftmontype=monitor;
123   var no:integer;
124       inparam,outparam:ioparam;
125       ASCII,NUMERIC:char;
126       MAIN:array[1..10] of char;
127       xleft:iodevice;
128       ready:boolean;
129       leftqueue:queue;
130
131
132   procedure entry setxleft(var dev:iodevice);
133   begin
134     xleft:=dev;
135     ready:=true;
136     continue(leftqueue);
137   end; {setxleft}
138
139   procedure entry getxleft(var dev:iodevice);
140   begin
141     if ready=false then delay(leftqueue);
142     dev:=xleft;
143     end; {getxleft}
144
145   procedure entry dataout(var comp,node:nametype;var outvalue:univ rstring);
146   begin
147     io(NUMERIC,outparam,xleft);
148     for no:=1 to namelength do io(comp[no],outparam,xleft);
149     for no:=1 to namelength do io(node[no],outparam,xleft);
150     for no:=1 to 4 do io(outvalue[no],outparam,xleft);
151     end;{dataout}
152
153   procedure entry asciiout(var comp:nametype;var ch:char);
154   begin
155     io(ASCII,outparam,xleft);
156     for no:=1 to namelength do io(comp[no],outparam,xleft);
157     io(ch,outparam,xleft);
158     end;{asciiout}
```

```
159 begin
160   inparam:=input;
161   outparam:=output;
162   ASCII:='a';
163   NUMERIC:='#';
164   MAIN:='MAIN';
165   ready:=false;
166   end!(outlefttype)
167
168
169 {*****}
170 * outrightmontype *
171 *****}
172
173 type outrightmontype=monitor;
174 var no:integer;
175 inparam;outparam:ioparam;
176 ASCII,NUMERIC:char;
177 MAIN:array[1..10] of char;
178 xright:iodevice;
179 ready:boolean;
180 rightqueue:queue;
181
182 procedure entry setxright(var dev:iodevice);
183 begin
184   xright:=dev;
185   ready:=true;
186   continue(rightqueue);
187 end; {setxright}
188
189 procedure entry getxright(var dev:iodevice);
190 begin
191   if ready=false then delay(rightqueue);
192   dev:=xright;
193   end; {getxright}
194
195 procedure entry dataout(var comp:node;nametype;var outvalue:univ rstring);
196 begin
197   io(NUMERIC,outparam,xright);
198   for no:=1 to namelength do io(comp[no],outparam,xright);
199   for no:=1 to namelength do io(node[no],outparam,xright);
200   for no:=1 to 4 do io(outvalue[no],outparam,xright);
201   end!dataout;
202
203 procedure entry asciiout(var comp;nametype;var ch:char);
204 begin
205   io(ASCII,outparam,xright);
206   for no:=1 to namelength do io(comp[no],outparam,xright);
207   io(ch,outparam,xright);
208   end!asciiout;
209 begin
210   inparam:=input;
211   outparam:=output;
```

```
212 ASCII:='a';
213 NUMERIC:='#';
214 MAIN:='MAIN' ;
215 ready:=false;
216 end;{outrighttype}
217
218
219 {*****}
220 * readterminalmontype *
221 {*****}
222
223 type readterminalmontype = monitor;
224 const LF='(10)'; del='(127)'; CR='(13)';
225 lineLength=133; lineLength1=134; {lineLength+1}
226 var space,backspace,linefeed:char;
227 inparam,outparam:ioParam;
228 lineArray[1..lineLength] of char;
229 pos:0..lineLength1;
230 tty:iodevice;
231
232 procedure entry read(var character:char);
233 var ch:char;
234 begin
235   if pos=0 then begin {read a new line}
236     repeat
237       io(ch,inparam,tty);
238       if ch()del then begin
239         io(ch,outparam,tty);
240         if ch='a' then if ch='z' then ch:=chr(ord(ch)-32);
241         if pos<lineLength then pos:=pos+1;
242         line[pos]:=ch end
243       else begin
244         if pos<>0 then begin
245           io(backspace,outparam,tty);
246           io(space,outparam,tty);
247           io(backspace,outparam,tty);
248           pos:=pos-1
249         end
250       end
251     until (ch=CR) or (pos=lineLength);
252     if ch=CR then begin
253       line[pos+1]:=LF;
254       io(linefeed,outparam,tty);
255     end;
256     pos:=0
257   end;
258   pos:=pos+1;
259   character:=line[pos];
260   if character=LF then pos:=0
261   end;{read}
262
263 begin {readterminalmontype}
264   tty:=s10;
```

```

265 inparam:=input;
266 outparam:=output;
267 space:= ' ';
268 backspace:=chr(8);
269 linefeed:=LF;
270 pos:=0;
271 endf{readterminalmontype}
272
273
274
275 {*****}
276 * writeterminalmontype *
277 {*****}
278 type writeterminalmontype = monitor;
279 var tty :iodevice;
280 outparam :ioparam;
281
282 procedure entry write(var ch:char);
283 begin
284   io(ch,outparam,tty)
285 endf{write}
286 begin
287   tty:=sio;
288   outparam:=output
289 endf{writeterminalmontype}
290
291
292 {*****}
293 * textwritemontype *
294 {*****}
295
296 type textwritemontype = monitor(name:mon;name:monitontype;
297 outleft:outleftmontype;
298 outright:outrightmontype);
299
300 procedure entry textwrite(var dest:name;var ch:char);
301 var answer :answertype;
302 begin
303   name:=mon.checkcomp(dest,answer);
304   case answer of
305     left: outleft.asciout(dest,ch);
306     right: outright.asciout(dest,ch)
307   endf{case}
308 endf
309 begin
310 endf{textwritemontype}
311
312 type filespectype=array[1..14] of char;
313 seqprogspectype=record
314   filename:filespectype;
315   stacktop:integer;
316   heaptop:integer;
317   startaddress:integer;

```



```

318 loadaddress:=integer;
319 end;
320 Jobprocesstype=(regulprocess,opcompprocess);
321
322 {*****}
323 * seqprogramstype *
324 {*****}
325
326 type seqprogramstype=monitor;
327 const regulfilename='DX1:REGUL3.REL';
328 regulspace=1000;
329 opcomfilename='DX1:OPCOM3.REL';
330 opcomspace=1000;
331 var regulfile,opcomfile:seqprogspectype;
332
333 procedure entry seqparms (var seqprogspec:seqprogspectype;
334 Jobprocess:jobprocesstype);
335 begin
336 case jobprocess of
337 regulprocess: seqprogspec:=regulfile;
338 opcompprocess: seqprogspec:=opcomfile
339 end (case);
340 end;{seqparms}
341
342 begin
343 regulfile.filename:=regulfilename;
344 regulfile.loadaddress:=0;
345 Loadseq(regulfile,regulspace);
346 opcomfile.filename:=opcomfilename;
347 opcomfile.loadaddress:=0;
348 Loadseq(opcomfile,opcomspace)
349 end; {seqprogramstype}
350
351 type address=record
352 name:nametype;
353 number:integer;
354 end;{address}
355 paramtypes=record
356 goalcomp:nametype;
357 intime,outtime,lasttime:integer;
358 tag:nodetype;
359 insig1,insig2,insig3,insig4,output:address;
360 scal1,scal2,scal3,scal4,level:real
361 end;{paramtype}
362 statetype=record
363 filterstate,yold,ipart,dpert:real
364 end;{statetype}
365 ddcnodetype=record
366 fwd,bwd:integer;
367 name:nametype;
368 priority:integer;
369 period,counter:integer;
370 outvalue:real;

```

```
371   parampart: paramtype;
372   state: statetype
373   end; { ddcnodetype }
374
375   {*****}
376   * nodemonitortype *
377   {*****}
378
379   const nodelistsize=20;
380   type nodemonitortype=monitor(
381     namemon: name;
382     monitortype: monitortype;
383     outleft: outleftmontype;
384     outright: outrightmontype;
385     inregindex: integer;
386     opqueue: queue;
387
388     procedure entry regputgetnode(
389       var node: ddcnodetype;
390       var anyfound: boolean);
391   begin
392     if regindex<>1 then begin
393       with nodelist[regindex] do begin
394         outvalue:=node.outvalue;
395         state:=node.state;
396         parampart.outtime:=node.parampart.outtime;
397         parampart.lasttime:=node.parampart.lasttime;
398       end {with}
399     end;
400     done:=false; anyfound:=false;
401     repeat
402       regindex:=nodelist[regindex].fwd;
403       if regindex=1 then done:=true
404     else begin
405       with nodelist[regindex] do begin
406         if period>0 then begin
407           counter:=counter-1;
408           if counter<=0 then begin
409             counter:=period;
410             anyfound:=true;
411             node:=nodelist[regindex];
412             done:=true;
413           end
414         end {with}
415       end
416     until done;
417     continue(opqueue)
418   end; { regputgetnode }
419
420   procedure entry getoutvalue(
421     var number: integer;
422     var outval: real);
423   begin
424     per:=nodelist[number].period;
```

```
424 with nodelist[number].parampart do
425   begin
426     outtime:=realttime;
427     if (outtime - intime)=(per div 2) then
428       lasttime:=outtime
429     end; {parampart}
430     outval:=nodelist[number].outvalue
431   end;{getoutvalue}
432
433
434
435
436 procedure lookup(var name:nametype; var ptr:integer);
437   begin
438     nodelist[1].name:=name;
439     ptr:=1;
440     repeat
441       ptr:=nodelist[ptr].fwd
442     until nodelist[ptr].name=name
443   end;{lookup}
444
445 procedure entry getrealttime(var rtime:integer);
446   begin
447     rtime:=realttime
448   end; {getrealttime}
449
450 procedure entry putoutvalue(var node:nametype;var value:univ real);
451   var ptr :integer;
452   begin
453     lookup(node,ptr);
454     if ptr=1 then {error(nonode)} else
455       if nodelist[ptr].parampart.tag() innode then {error(noinnode)} else
456         begin
457           nodelist[ptr].outvalue:=value;
458           nodelist[ptr].parampart.intime:=realttime
459         end;
460       end;{putoutvalue}
461
462 procedure entry opgetnode(var node:ddcnodetype;
463   var notfound,notspace:boolean);
464   var ptr:integer;
465   begin
466     lookup(node.name,ptr);
467     if ptr() then begin
468       notfound:=false;
469       node:=nodelist[ptr] end
470     else begin
471       notfound:=true;
472       notspace:=nodelist[2].fwd=2
473     end
474   end;{opgetnode}
475
476 procedure entry delete(var nodename:nametype;var result:integer);
477   var this,ptr:integer;
478   begin
```

```
477 result:=0;
478 lookup(nodename,this);
479 if this=1 then result:=1
480 else begin
481   ptr:=odelist[this].fwd;
482   while (ptr<>1) and (result<>2) do begin
483     with node1st[ptr].parampart do begin
484       if insig1.name=nodename then result:=2;
485       if tag<> outnode then begin
486         if insig2.name=nodename then result:=2;
487         if tag=outnode then begin
488           if insig3.name=nodename then result:=2;
489           if insig4.name=nodename then result:=2;
490         end
491       end
492     end;
493     ptr:=odelist[ptr].fwd
494   end{while}
495 end;
496 if result=0 then begin
497   if this=regindex then delay(opqueue);
498   with node1st[this] do begin
499     node1st[fwd].bwd:=bwd;
500     node1st[bwd].fwd:=fwd;
501     bwd:=2;
502     fwd:=odelist[2].fwd;
503     node1st[2].fwd:=this;
504     node1st[fwd].bwd:=this;
505   end{with}
506 end
507 end{delete}
508
509 procedure entry link(var nodeidcnodetype);
510 var this,ptr:integer;
511 begin
512   lookup(node.name,this);
513   if this<>1 then begin
514     with node1st[this] do begin
515       period:=node.period;
516       counter:=counter+node.counter;
517       parampart:=node.parampart
518     end {with} end
519   else begin {a new node}
520     this:=odelist[2].fwd;
521     ptr:=odelist[this].fwd;
522     node1st[ptr].bwd:=2;
523     node1st[2].fwd:=ptr;
524     node1st[this]:=node;
525     ptr:=1;
526     with node1st[this] do begin
527       repeat
528         ptr:=odelist[ptr].fwd
529       until priority<node1st[ptr].priority;
```

```

530   fwd:=ptr;
531   bwd:=odelist[ptr].bwd;
532   nodelist[ptr].bwd:=this;
533   nodelist[bwd].fwd:=this
534   end {with}
535   end
536   end:{link}
537
538   procedure entry checkname(var name:nametype;
539                               var ptr,pri:integer);
540   begin
541     lookup(name:ptr);
542     if ptr() then pri:=nodelist[ptr].priority
543     end:{checkname}
544
545   procedure entry datawrite(var comp,node:nametype;var outvalue:real);
546   var ptr :integer;
547     answer :answertype;
548   begin
549     name:=comp(comp,answer);
550     case answer of
551     left:   outleft.dataout(comp,node,outvalue);
552     right:  outhright.dataout(comp,node,outvalue);
553     myself: {error(longway)};
554     nonexistent: {error(nocomp)};
555     end:{case}
556     end:{datawrite}
557
558   begin {nodemonitortype}
559     regindex:=1;
560     with nodelist[i] do begin
561       fwd:=1; bwd:=1;
562       priority:=32767;
563     end{with}
564     for i:=2 to nodelistsize do begin
565       with nodelist[i] do begin
566         fwd:=i+1; bwd:=i-1
567       end {with}
568     end{for}
569     nodelist[2].bwd:=nodelistsize;
570     nodelist[nodelistsize].fwd:=2
571     end{nodemonitortype}
572
573   {*****}
574   * timetable *
575   {*****}
576
577   type timetabletype=monitor;
578   var lags:integer;
579     regqueue:queue;
580   procedure entry wait;
581   begin
582     if lags=0 then delay(regqueue);

```

```

583   lags:=lags-1
584   endif{wait}
585
586   procedure entry schedule;
587   begin
588     lags:=lags+1;
589     if lags>10 then begin
590       lags:=10;
591       {alarm}
592     end;
593     continue{regqueue}
594   endifschedule}
595
596   begin
597     lags:=0
598   endif{metabletype}
599
600   {*****}
601   * inlefttype *
602   {*****}
603
604
605   inlefttype:=process(nodemonitorotype;namemon;namemonitorotype;
606                      buffmon;buffmonotype;outright;outrightmontype;
607                      outleft;outleftmontype;
608                      writeterminalmon;writeterminalmontype);
609
610   const priority=1;
611   var xfertype,ch
612       answer
613       no,nr
614       value
615       comp,myname,node,MAIN
616       inparam,outparam
617       xleft
618       error
619
620   begin
621     setpri(priority);
622     inparam:=input;
623     outparam:=output;
624     outleft:=getxleft(xleft);
625     namemon:=getmyname(myname);
626     MAIN:=MAIN
627     error[1]:=myname;
628     error[2]:=inl err  ;
629     cycle
630     io(xfertype,inparam,xleft);
631     while (xfertype{.a}') and (xfertype{.}') do io(xfertype,inparam,xleft);
632     for no:=1 to namelength do io(comp[no],inparam,xleft);
633     if xfertype='a' then io(ch,inparam,xleft)
634     else begin
635       if xfertype='#' then
636         begin

```

```

636 for no:=1 to namelength do io(node[no],inparam,x:left);
637 for no:=1 to 4 do io(value[no],inparam,x:left);
638 end
639 else
640 begin
641   namemon.checkcomp(MAIN,answer);
642   case answer of
643     left:
644       for nr:=1 to 2 do
645         outleft.asciout(MAIN,error[no,nr]);
646       for nr:=1 to 2 do
647         outright.asciout(MAIN,error[no,nr]);
648       for nr:=1 to 2 do
649         myself:
650           writeterminalmon.write(error[no,nr])
651         end; {case}
652       end
653     end;
654   end;
655   namemon.checkcomp(comp,answer);
656   case answer of
657     {error(circuiss)};
658     nonexistent: {error(nocomp)};
659     right:
660       if x.fertype='g' then
661         outright.asciout(comp,ch) else
662         outright.dataout(comp,node,value);
663       if x.fertype='a' then
664         if myname='MAIN' then writeterminalmon.write(ch)
665         else namemon.putoutvalue(node,value)
666       end; {case}
667     end; {cycle}
668   end; {inlefttype}
669 {*****}
670 * inrighttype *
671 {*****}
672
673 inrighttype=process(node:mon;nodemon;nodemonitortype;namemon;namemonitortype;
674   buffmon;buffmontype;outleft:outleftmontype;
675   outright:outrightmontype;
676   writeterminalmon:writeterminalmontype);
677
678 const priority=1;
679 var x.fertypesch
680   answer
681   no,nr
682   value
683   comp;myname,node,MAIN
684   inparam,outparam
685   xright
686   error
687
688 begin

```

```

689 setpri(priority);
690 inparam:=input;
691 outparam:=output;
692 outright.getxright(xright);
693 namemon.getmyname(myname);
694 MAIN:= 'MAIN
695 error[1]:=myname;
696 error[2]:= 'inR err
697 cycle
698 io(xfertype,inparam,xright);
699 while (xfertype='a') and (xfertype!='#') do io(xfertype,inparam,xright);
700 for no:=1 to namelength do io(comp[no],inparam,xright);
701 if xfertype='a' then io(ch,inparam,xright)
702 else begin
703   if xfertype='#' then
704     begin
705       for no:=1 to namelength do io(node[no],inparam,xright);
706       for no:=1 to 4 do io(value[no],inparam,xright);
707     end
708   else
709     begin
710       namemon.checkcomp(MAIN,answer);
711       case answer of
712         left:   for no:=1 to 2 do
713                   outfile.asciiout(MAIN,error[no,nr]);
714                   for no:=1 to 2 do
715                   outfile.asciiout(MAIN,error[no,nr]);
716                   for no:=1 to 2 do
717                   outfile.asciiout(MAIN,error[no,nr]);
718                   for no:=1 to 2 do
719                   writeterminalmon.write(error[no,nr]);
720                   writeterminalmon.write(error[no,nr]);
721                 end; {case}
722               end;
723             case answer of
724               {error(circmess)};
725               nonexistent: {error(nocomp)};
726               left:        if xfertype='a' then
727                             outfile.asciiout(comp,node,value);
728                             outfile.asciiout(comp,ich) else
729                             if xfertype='a' then
730                               if myname='MAIN' then writeterminalmon.write(ch)
731                               else buffmon.write(ch)
732                             else nodemon.putoutvalue(node,value)
733                             end;{case}
734                             end;{cycle}
735                           end;{inrighttype}
736                         {*****}
737                         * clocktype *
738                         *****}
739                       *****}
740                       *****}
741                       *****}

```



```

742 type clocktype=process(timetable:timetabletype);
743 const priority=1;
744     maxtime=$2767;
745     var nexttime:integer;
746     begin
747         setpri(priority);
748         nexttime:=realtime;
749         cycle
750         timetable.schedule;
751         if nexttime=maxtime then nexttime:=0
752         else nexttime:=nexttime+1;
753         sleep(nexttime)
754         end {cycle}
755     end {clocktype}
756
757 {*****}
758 * regulator type *
759 {*****}
760
761 regulator type=process(buffmon:buffmontype;
762     readterminalmon:readterminalmontype;
763     writeterminalmon:writeterminalmontype;
764     seqprograms:seqprogramstype;
765     nodemon:nodemontortype;
766     timetable:timetabletype;
767     namemon:namemontortype;
768     outleft:outleftmontype;
769     outright:outrightmontype);
770
771
772 const priority=2;
773 var regufile:seqprogspectype;
774     MAIN :nametype;
775 program regulator(regufile:seqprogspectype);
776 entry read,write,wait;
777     regputgetnode,getoutvalue,getrealtime,datawrite;
778     getmyname;
779
780 procedure entry read(var character:char);
781 var name:nametype;
782     begin
783         namemon.getmyname(name);
784         if name=MAIN then readterminalmon.read(character)
785         else buffmon.read(character)
786     end;
787 procedure entry write(var character:char);
788 var answer :answertype;
789     name :nametype;
790     begin
791         namemon.checkcomp(MAIN,answer);
792         case answer of
793             myself: writeterminalmon.write(character);
794             left: outleft.asciiout(MAIN,character);

```

```
795 right: outright.asciiout(MAIN,character)
796 end(case)
797 end(write)
798 procedure entry wait;
799 begin
800 timetable.wait
801 end;
802 procedure entry regputgetnode(var node:ddcnodeType;
803                               var anyfound:boolean);
804 begin
805   nodemon.regputgetnode(node,anyfound)
806 end;
807 procedure entry getoutvalue(var number:integer;
808                              var outvalue:real);
809 begin
810   nodemon.getoutvalue(number,outvalue)
811 end;
812 procedure entry getrealtime(var rtime:integer);
813 begin
814   nodemon.getrealtime(rtime)
815 end;
816 procedure entry datawrite(var comp,nodename:nametype;
817                             var sendvalue:real);
818 var answer :answertype;
819 begin
820   namemon.checkcomp(comp,answer);
821 case answer of
822   left:   outright.dataout(comp,nodename,sendvalue);
823   right:  outright.dataout(comp,nodename,sendvalue);
824   myself: {error(longway)};
825   nonexistent: {error(nocomp)}
826 end(case)
827 end;
828 procedure entry getmyname(var thisname:nametype);
829 begin
830   namemon.getmyname(thisname)
831 end;
832 procedure entry setpri(priority);
833 begin
834   MAIN:=MAIN
835   seqprograms.seqparms(regulfile,regulprocess);
836   regulator(regulfile)
837 end;
838 {*****}
839 * opcomtype *
840 {*****}
841
842
843
844
845
846
847
```

```

848 opcomtype:=process(buffmon:buffmontype;
849   readterminalmon:readterminalmontype;
850   writeterminalmon:writeterminalmontype;
851   textwriteon:texwriteontype;
852   seqprograms:seqprogramstype;
853   nodemon:nodemonitortype;
854   namemon:namemonitortype;
855   outleft:outleftmontype;
856   outright:outrightmontype);
857 const priority:=3; LF:=('10'); CR:=('13');
858 var opcomfile:seqprogspectype;
859   MAIN
860   :nametype;
861 program opcom(opcomfile:seqprogspectype);
862 entry read,write,opgetnode,delete,
863   link,checkname,setwname,
864   setleftnames,setrightnames,
865   checkcomp,texwrite;setxleft;setxright;
866 procedure entry read(var character:char);
867   var name :nametype;
868   answer:answertype;
869   ch:char;
870 begin
871   namemon.getwname(name);
872   if name=MAIN then readterminalmon.read(character)
873   else
874     begin
875       buffmon.read(character);
876       namemon.checkcomp(MAIN,answer);
877       case answer of
878         left:
879           begin
880             outleft.asciout(MAIN,character);
881             if character=LF then begin
882               ch:=CR;
883               outleft.asciout(MAIN,ch)
884             end
885           end;
886         right:
887           begin
888             outright.asciout(MAIN,character);
889             if character=LF then begin
890               ch:=CR;
891               outright.asciout(MAIN,ch);
892             end
893           end;
894         {case}
895       end;
896     end;
897   procedure entry write(var character:char);
898   var answer :answertype;
899   name :nametype;
900   begin
901     namemon.checkcomp(MAIN,answer);
902     case answer of

```

```
901 myself: writeterminalmon.write(character);
902 left:  outleft.asciiout(MAIN,character);
903 right:  outright.asciiout(MAIN,character)
904     end(case)
905 end(ifwrite)
906 procedure entry opgetnode(var node:ddcnodetype;
907     var notfound,notspace:boolean);
908 begin
909     nodemon.opgetnode(node,notfound,notspace)
910 end;
911 procedure entry link(var node:ddcnodetype);
912 begin
913     nodemon.link(node)
914 end;
915 procedure entry delete(var nodename:nametype;
916     var result:integer);
917 begin
918     nodemon.delete(nodename,result)
919 end;
920 procedure entry checkname(var name:nametype;
921     var ptr,pri:integer);
922 begin
923     nodemon.checkname(name,ptr,pri)
924 end;
925 procedure entry textwrite(var dest:nametype;
926     var ch:char);
927 begin
928     textwritemon.textwrite(dest,ch);
929 end;
930 procedure entry checkcomp(var thisname:nametype;
931     var answer:answertype);
932 begin
933     namemon.checkcomp(thisname,answer);
934 end;
935 procedure entry setmyname(var myname:nametype);
936 begin
937     namemon.setmyname(myname);
938 end;
939 procedure entry setleftnames(var thisname:nametype;
940     var i:integer);
941 begin
942     namemon.setleftnames(thisname,i);
943 end;
944 procedure entry setrightnames(var thisname:nametype;
945     var i:integer);
946 begin
947     namemon.setrightnames(thisname,i);
948 end;
949 procedure entry setxleft(var dev:iodevice);
950 begin
951     outleft.setxleft(dev);
952 end;
953 procedure entry setxright(var dev:iodevice);
```

```

954 begin
955   outright.setxright(dev);
956 end;
957
958 begin
959   setpri(priority);
960   MAIN:=MAIN
961   seqprograms.seqparms(opcomfile,opcomprocess);
962   opcom(opcomfile)
963 end;opcomtype}
964
965
966 {*****
967 * main *
968 *****}
969
970 var readterminalmon:readterminalmontype;
971     writeterminalmon:writeterminalmontype;
972     textwritemon:textwritemontype;
973     seqprograms:seqprogramstypetype;
974     nodemon:nodemonitortype;
975     timetable:timetabletype;
976     clock:clocktype;
977     regulator:regulatorontype;
978     opcom:opcomtype;
979     namemon:namemonitortype;
980     buffmon:buffmontype;
981     outfile:outfilemontype;
982     outright:outrightmontype;
983     inleft:inlefttype;
984     inright:inrighttype;
985
986 begin
987   init seqprograms,timetable,buffmon,
988     outfile,outright,namemon,
989     readterminalmon,
990     writeterminalmon,
991     textwritemon(namemon,outleft,outright),
992     nodemon(namemon,outleft,outright),
993     clock(timetable),
994     regulator(buffmon,readterminalmon,writeterminalmon,seqprograms,
995     nodemon,timetable,namemon,outleft,outright),
996     opcom(buffmon,readterminalmon,writeterminalmon,textwritemon,
997     seqprograms,nodemon,outleft,outright),
998     inleft(nodemon,namemon,buffmon,outleft,outright,writeterminalmon),
999     inright(nodemon,namemon,buffmon,outleft,outright,writeterminalmon)
1000
1001 end.

```

ERRORS DETECTED: 0

LINE STMT LEVEL NEST SOURCE STATEMENT

```

1  program opcom;
2  label 999;
3  type entryprocedures=(opgetnode1,delete1,link1,checkname1,setmyname1,
4    setleftname1,setrightname1,checkcomp1,
5    textwritel,setxleft1,setxright1);
6  procedure entprocedure(entryprocedure:entryprocedures); external;
7
8  const maxcomputers=4;
9    tickspersecond=50.0;
10   blanks=' ';;
11   AD='AD ';;
12   DA='DA ';;
13   MAIN='MAIN ';;
14   EXT='EXT ';;
15   type nametype=array[1..10] of char;
16   address=record
17     name:nametype;
18     number:integer
19   end;faddress;
20   nodetype=(inode,PIDnode,OUTnode);
21   paramtype=record
22     goalcomp:nametype;
23     intime,outtime,lasttime:integer;
24     case tag:nodetype of
25       inode:(insig,outplace,address;
26         scale,filter:real);
27       PIDnode:(PIDin,PIDref,PIDout,address;
28         k,tit,d,alfa,beta,limit:real);
29       outnode:(insig1,insig2,insig3,
30         insig4,output:address;
31         scal1,scal2,scal3,scal4,level:real)
32     end;fparamtype;
33   statetype=record
34     filterstate,yold,ipart,dpert:real
35   end;fstatetype;
36   ddcnodetype=record
37     fwd,bwd:integer;
38     name:nametype;
39     priority:integer;
40     period,counter:integer;
41     outvalue:real;
42     parampart:paramtype;
43     state:statetype
44   end;fddcnodetype;
45   commandtype=array[1..9] of char;
46   opindex=(openx,showx,linkx,delelex,lastopindex);
47   variableindex=(goalcomp,periodx,syncx,insig,outplacex,scalx,
48     filterx,PIDinx,PIDrefx,PIDoutx,kx,tix,
49     tdx,limitx,insig1x,insig2x,
50     insig3x,insig4x,outputx,scalix,
51     scal2x,scal3x,scal4x,
52     levelx,lastvarindex);
53   errors=(fewarg,toolmanyarg,illname,nospace,
54     notopen,blankadd,nonode,referr,ADDAerr;
```

LINE STMT LEVEL NEST SOURCE STATEMENT

```

55 noname:prierr,noop,tager,illcomp,desterr);
56 texttype=array[1..132] of char;
57 answer=(right,left,myself,nonexistent);
58 iodevice=(s10,s11,s12);
59
60 var op:array[opindex] of commandtype;
61 vars:array[variableindex] of commandtype;
62 opx:opindex;
63 varindex:variableindex;
64 node:ddcnodetype;
65 name:nametype;
66 command:commandtype;
67 opened:boolean;
68 myname:nametype;
69 actcomp:nametype;
70 i,j:integer;
71 exist:answertype;
72 buff:texttype;
73 ch:char;
74 LF:char;
75
76 procedure opgetnode(var node:ddcnodetype;
77 var notfound,notspace:boolean);
78
79 begin
80   entprocedure(opgetnode);
81 end;
82 procedure delete2(var name:nametype;
83 var result:integer);
84 begin
85   entprocedure(delete);
86 end;
87 procedure link2(var node:ddcnodetype);
88 begin
89   entprocedure(link);
90 end;
91 procedure checkname(var name:nametype;
92 var nr,pr:integer);
93 begin
94   entprocedure(checkname);
95 end;
96 procedure setmyname(var myname:nametype);
97 begin
98   entprocedure(setmyname);
99 end;
100 procedure setleftnames(var thisname:nametype;
101 var i:integer);
102 begin
103   entprocedure(setleftnames);
104 end;
105 procedure setrightnames(var thisname:nametype;
106 var i:integer);
107 begin
108   entprocedure(setrightnames);
109 end;

```

LINE STMT LEVEL NEST SOURCE STATEMENT

```

109 procedure checkcomp(var thisname:nametype;
110 var answer:answertype);
111 begin
112   entprocedure(checkcomp1)
113 end;
114 procedure textwrite(var dest:nametype;
115 var ch1char);
116 begin
117   entprocedure(textwritel)
118 end;
119 procedure setxleft(var des:iodevice);
120 begin
121   entprocedure(setxleft1)
122 end;
123 procedure setxright(var des:iodevice);
124 begin
125   entprocedure(setxright1)
126 end;
127
128 procedure error(err:errors);
129 begin
130   case err of
131     fewarg:   writeln('missing arguments');
132     toomanyarg: writeln('too many arguments');
133     illname:  writeln('illegal name');
134     nospace:  writeln('the nodelist is full');
135     notopen:  writeln('no node is open');
136     blankaddr: writeln('undefined signal');
137     nonode:   writeln('the node didn't exist');
138     referr:   writeln('other nodes use this node');
139     ADDAerr:  writeln('please don't send to AD-converter ',
140                      'or read from DA-converter');
141     noname:   writeln('undefined signal');
142     prierr:   writeln('the value is not ',
143                      'available here');
144     noop:     writeln('illegal operation');
145     tagerr:   writeln('wrong nodetype');
146     illcomp:  writeln('no computer name');
147     desterr:  writeln('don't send to DA-converter ',
148                      'of another computer,stupid!');
149   end;
150   readln;
151   write('myname,');
152   goto 999;
153 end;
154
155 procedure writeaddress(addr:address);
156 begin
157   with addr do begin
158     if name=AD then writeln('AD ',number:2)
159     else if name=DA then writeln('DA ',number:2)
160     else writeln(name)
161   end {with};
162   end;

```


LINE STMT LEVEL NEST SOURCE STATEMENT

```

163 procedure readname;
164 begin
165     if eoln then error('fewarg');
166     read(name);
167     if not eoln then error('toomanyarg');
168     if (name=AD) or (name=DA) or
169        (name=blanks) then error('illname')
170     end;
171     readname;
172
173 procedure open;
174 var notfound,notspace,done:boolean;
175     nodestring:array[1..8] of char;
176     procedure initname(var addr:address);
177     begin
178         addr.name:=blanks;
179         addr.number:=0
180     end;
181     initname;
182     opened:=false;
183     readname;
184     node.name:=name;
185     opgetnode(node,notfound,notspace);
186     if notfound then begin
187         if notspace then error('nospace')
188         else begin
189             with node do begin
190                 repeat
191                     write('* priority= ');
192                     readln(priority);
193                     until (priority<0) and (priority<32767);
194                     period:=0;
195                     counter:=1;
196                     outvalue:=0.0
197                 end;
198                 with node.parampart do begin
199                     repeat
200                         done:=true;
201                         write('* nodetype= ');
202                         read(nodestring);
203                         if nodestring='INNODE' then
204                             tag:=innode
205                         else if nodestring='PIDNODE' then
206                             tag:=PIDnode
207                         else if nodestring='OUTNODE' then
208                             tag:=outnode
209                         else begin done:=false; readln end
210                     until done;
211                     goalcomp:=myname;
212                     case tag of
213                         innode: begin
214                             initname(inside);
215                             scale:=1.0;
216                             filter:=1.0;

```

LINE STMT LEVEL NEST SOURCE STATEMENT

```

217 40 2 9          initname(outplace);
218 41 2 9          intime:=0;
219 42 2 9          outtime:=0;
220 43 2 9          lasttime:=0;
221 44 2 9          end;
222 45 2 8          PIDnode:begin
223 46 2 9              limit:=1.0;
224 47 2 9              k:=0.0;
225 48 2 9              ti:=0.0;
226 49 2 9              td:=0.0;
227 50 2 9          initname(PIDin);
228 51 2 9          initname(PIDref);
229 52 2 9          initname(PIDout)
230          end;
231 53 2 8          outnode:begin
232 54 2 9              initname(insig1); scal1:=0.0;
233 56 2 9              initname(insig2); scal2:=0.0;
234 58 2 9              initname(insig3); scal3:=0.0;
235 60 2 9              initname(insig4); scal4:=0.0;
236 62 2 9              initname(output);
237 63 2 9              level:=0.0
238          end
239          end {case}
240          end {with}
241          with node.state do begin
242 64 2 7              filterstate:=0.0;
243 66 2 7              yold:=0.0;
244 68 2 7              ipart:=0.0;
245 69 2 7              dpart:=0.0;
246 70 2 7              end {with}
247          end
248          end;
249          opened:=true;
250 72 2 1          node.counter:=0;
251 73 2 1          end {open}
252
253          procedure show;
254          begin
255 1          if not opened then error(notopen);
256 3          with node do begin
257 5              write('NODENAME: '); writein(name);
258 7              write('PERIOD: '); writein(period);
259 9              write('PRIORITY: '); writein(priority);
260 11             write('OUTVALUE: '); writein(outvalue);
261 13             end {with}
262 14             with node.parampart,node.state do begin
263 16                 case tag of
264 17                     5: innode: begin
265 18                         write('INSIG '); writeaddress(insig);
266 20                         write('FILTER '); writein(filter);
267 22                         write('SCALE '); writein(scale);
268 24                         write('FILTERSTATE: '); writein(filterstate);
269 26                         write('GOALCOMP: '); writein(goalcomp);
270 28                         write('OUTPLACE: '); writeaddress(outplace);

```

LINE STMT LEVEL NEST SOURCE STATEMENT

```

271 30 2 6 write('INTIME : '); writeln(intime);
272 32 2 6 write('OUTTIME : '); writeln(outtime);
273 34 2 6 write('LASTIME: '); writeln(lasttime);
274 36 2 6 end;
275 37 2 5 PIDnode:begin
276 38 2 6 write('PIDREF : '); writeaddress(PIDref);
277 40 2 6 write('PIDIN : '); writeaddress(PIDin);
278 42 2 6 write('PIDOUT : '); writeaddress(PIDout);
279 44 2 6 write('GOALCOMP: '); writeln(goalcomp);
280 46 2 6 write('K : '); writeln(K);
281 48 2 6 write('TI : '); writeln(TI);
282 50 2 6 write('TD : '); writeln(TD);
283 52 2 6 write('LIMIT : '); writeln(limit);
284 54 2 6 write('YOLD : '); writeln(yold);
285 56 2 6 write('IPART : '); writeln(ipart);
286 58 2 6 write('DPART : '); writeln(dpart);
287 60 2 6 end;
288 61 2 5 outnode:begin
289 62 2 6 write('INSIG1 : '); writeaddress(insig1);
290 64 2 6 write('SCAL1 : '); writeln(scal1);
291 66 2 6 write('INSIG2 : '); writeaddress(insig2);
292 68 2 6 write('SCAL2 : '); writeln(scal2);
293 70 2 6 write('INSIG3 : '); writeaddress(insig3);
294 72 2 6 write('SCAL3 : '); writeln(scal3);
295 74 2 6 write('INSIG4 : '); writeaddress(insig4);
296 76 2 6 write('SCAL4 : '); writeln(scal4);
297 78 2 6 write('OUTPUT : '); writeaddress(output);
298 80 2 6 write('GOALCOMP: '); writeln(goalcomp);
299 82 2 6 write('LEVEL : '); writeln(level);
300 84 2 6 end
301 end {case}
302 end {with}
303 end{show}
304
305 procedure link;
306 var ts:real;
307 begin
308 1 2 if not opened then error(notopen);
309 3 2 with node,parampart do begin
310 5 2 case tag of
311 6 2 innode: begin
312 7 2 5 if (myname()goalcomp) and (outplace.name=DA)
313 8 2 6 then error(desterr);
314 9 2 5 if insig.name=blanks then error(blankaddr)
315 end;
316 PIDnode:begin
317 11 2 4 if (myname()goalcomp) and (PIDout.name=DA)
318 12 2 5 then error(desterr);
319 13 2 6 if PIDref.name=blanks then
320 15 2 6 error(blankaddr);
321 16 2 5 if PIDin.name=blanks then
322 17 2 6 error(blankaddr);
323 18 2 5 ts:=node.period/tickspersecond;
324 19 2 5 if ti>0.0 then alfa:=k*ts/ti

```

LINE	STMT	LEVEL	NEST	SOURCE STATEMENT
325	21	2	6	else alpha:=0.0;
326	22	2	5	if (ts>0.0) and (td>0.0) then
327	23	2	6	beta:=k*td/ts
328	24	2	6	else beta:=0.0
329				end;
330	25	2	4	outnode:begin
331	26	2	5	if (myname()goalcomp) and (output.name=DA)
332	27	2	6	then error(deserr);
333	28	2	5	if (scal1<>0.0) and
334				(insig1.name=blanks) then
335	29	2	6	error(blankaddr);
336	30	2	5	if (scal2<>0.0) and
337				(insig2.name=blanks) then
338	31	2	6	error(blankaddr);
339	32	2	5	if (scal3<>0.0) and
340				(insig3.name=blanks) then
341	33	2	6	error(blankaddr);
342	34	2	5	if (scal4<>0.0) and
343				(insig4.name=blanks) then
344	35	2	6	error(blankaddr)
345				end
346				end(case)
347				end(with)
348	36	2	1	link2(node)
349				end(link)
350				end(link)
351				end(link)
352				end(link)
353				end(link)
354	1	2	1	readname;
355	2	2	1	delete2(name,result);
356	3	2	1	if result=1 then error(nonode);
357	5	2	1	if result=2 then error(referr)
358				end(delete)
359				end(delete)
360				end(delete)
361				end(delete)
362				end(delete)
363				end(delete)
364	1	2	1	var r:real;
365	2	2	2	begin
366	3	2	2	if node.parampart.tag()tag then
367	5	2	1	error(tagerr);
368	6	2	1	read(r);
369	8	2	1	if not eoln then error(toomanyarg);
370				variable:=r
371				end;
372				end;
373				end;
374				end;
375	1	2	1	procedure readint(var variable:integer);
376	3	2	1	begin
377	4	2	1	if eoln then error(fewarg);
378	6	2	1	read(i);
				if not eoln then error(toomanyarg);
				variable:=i;
				end;

LINE	STMT	LEVEL	NEST	SOURCE STATEMENT
379	7	2	1	end;
380				
381				procedure readaddr(var signal:address;
382				tag:nodetype;
383				outsignal:boolean);
384				var nr,pri:integer;
385				begin
386	1	2	1	if node.parampart.tag()tag then
387	2	2	2	error(tager);
388	3	2	1	if eoln then error(fewarg);
389	5	2	1	read(name);
390	6	2	1	if name()blanks then begin
391	8	2	3	if outsignal and (name=AD) then
392	9	2	4	error(ADaerr);
393	10	2	3	error(ADaerr);
394	11	2	4	if (not outsignal) and (name=DA) then
395	12	2	3	error(ADaerr);
396	14	2	3	if (name=DA) or (name=AD) then begin
397	16	2	5	if eoln then error(fewarg);
398	17	2	5	read(nr) end
399	19	2	4	else if (name()EXT) then begin
400	21	2	6	. if node.parampart.goalcomp=myname then begin
401	22	2	8	checkname(name,nr,pri);
402	24	2	8	if nr=1 then error(noname);
403				if node.priority(=pri then error(prierr) end
404				end
405	26	2	1	end;
406	28	2	1	- if not eoln then error(toomanyarg);
407	29	2	1	signal.name:=name;
408				signal.number:=nr
409				end!(readaddr);
410				procedure readcomp(var comp:nametype);
411				begin
412	1	2	1	if eoln then error(fewarg);
413	3	2	1	read(comp);
414	4	2	1	if not eoln then error(toomanyarg);
415	6	2	1	checkcomp(comp,exist);
416	7	2	1	if exist=nonexistent then error(illcomp)
417				end;
418				
419				procedure setvariable;
420				begin
421	1	2	1	with node.parampart do begin
422	3	2	3	varindex:=goalcomp;
423	4	2	3	varslastvarindex:=command;
424	5	2	3	while varslvarindex()command do
425	6	2	4	varindex:=succ(varindex);
426	7	2	3	if not opened then error(notopen);
427	9	2	3	case varindex of
428	10	2	4	goalcompx: readcomp(goalcomp);
429	11	2	4	periodx: readint(node.period);
430	12	2	4	syncx: readint(node.counter);
431	13	2	4	insigx: readaddr(insig;innode,false);
432	14	2	4	outplacex: readaddr(outplace;innode,true);

LINE	STMT	LEVEL	NEST	SOURCE STATEMENT
433	15	2	4	scalex: readreal(scale,inode);
434	16	2	4	filterx: readreal(filter,inode);
435	17	2	4	PIDinfx: readaddr(PIDin,PIDnode,false);
436	18	2	4	PIDrefx: readaddr(PIDref,PIDnode,false);
437	19	2	4	PIDoutfx: readaddr(PIDout,PIDnode,true);
438	20	2	4	Kx: readreal(k,PIDnode);
439	21	2	4	tx: readreal(tx,PIDnode);
440	22	2	4	tdx: readreal(td,PIDnode);
441	23	2	4	limitx: readreal(limit,PIDnode);
442	24	2	4	insig1x: readaddr(insig1,outnode,false);
443	25	2	4	insig2x: readaddr(insig2,outnode,false);
444	26	2	4	insig3x: readaddr(insig3,outnode,false);
445	27	2	4	insig4x: readaddr(insig4,outnode,false);
446	28	2	4	outputx: readaddr(output,outnode,true);
447	29	2	4	scal1x: readreal(scal1,outnode);
448	30	2	4	scal2x: readreal(scal2,outnode);
449	31	2	4	scal3x: readreal(scal3,outnode);
450	32	2	4	scal4x: readreal(scal4,outnode);
451	33	2	4	levelx: readreal(level,outnode);
452	34	2	4	lastvarindex: error(noop)
453				end (case)
454				end (with)
455				end(setvariable)
456				
457				procedure initialize;
458				var done:boolean;
459				i:integer;
460				xleft,xright:iodevice;
461				begin
462	1	2	1	LF:=chr(10);
463	2	2	1	optopenx:=
464	3	2	1	optshowx:=
465	4	2	1	optlinkx:=
466	5	2	1	optdeletex:=
467	6	2	1	vars[periodx]:=
468	7	2	1	vars[syncx]:=
469	8	2	1	vars[insigx]:=
470	9	2	1	vars[outputx]:=
471	10	2	1	vars[scalx]:=
472	11	2	1	vars[filterx]:=
473	12	2	1	vars[goalcompx]:=
474	13	2	1	vars[PIDinfx]:=
475	14	2	1	vars[PIDrefx]:=
476	15	2	1	vars[PIDoutfx]:=
477	16	2	1	vars[kx]:=
478	17	2	1	vars[tx]:=
479	18	2	1	vars[tdx]:=
480	19	2	1	vars[limitx]:=
481	20	2	1	vars[insig1x]:=
482	21	2	1	vars[insig2x]:=
483	22	2	1	vars[insig3x]:=
484	23	2	1	vars[insig4x]:=
485	24	2	1	vars[outputx]:=
486	25	2	1	vars[scal1x]:=

LINE	STMT	LEVEL	NEST	SOURCE STATEMENT
487	26	2	1	vars[scal2x]:= 'SCAL2 ';
488	27	2	1	vars[scal3x]:= 'SCAL3 ';
489	28	2	1	vars[scal4x]:= 'SCAL4 ';
490	29	2	1	vars[level1x]:= 'LEVEL ';
491	30	2	1	opened:=false;
492	31	2	1	actcomp:= 'MAIN ';
493	32	2	1	done:=false; i:=1;
494	34	2	1	repeat
495	35	2	2	write('NAME OF LEFT COMPUTER ',i:2,'');
496	36	2	2	readln(name);
497	37	2	2	if name=blanks then done:=true
498	39	2	3	else setleftnames(name,i);
499	40	2	2	i:=i+1;
500	41	2	2	until done or (i>maxcomputers);
501	42	2	1	done:=false; i:=1;
502	44	2	1	repeat
503	45	2	2	write('NAME OF RIGHT COMPUTER ',i:2,'');
504	46	2	2	readln(name);
505	47	2	2	if name=blanks then done:=true
506	49	2	3	else setrightnames(name,i);
507	50	2	2	i:=i+1;
508	51	2	2	until done or (i>maxcomputers);
509	52	2	1	done:=false;
510	53	2	1	repeat
511	54	2	2	write('NAME OF THIS COMPUTER:');
512	55	2	2	readln(myname);
513	56	2	2	if myname()=blanks then done:=true;
514	58	2	2	until done;
515	59	2	1	setmyname(myname);
516	60	2	1	if myname()=MAIN then begin
517	62	2	3	xleft:=s10;
518	63	2	3	xright:=s11
519				end
520	64	2	2	else begin
521	65	2	3	xleft:=s11;
522	66	2	3	xright:=s12;
523	67	2	3	write(myname,'');
524				end;
525	68	2	1	setxleft(xleft);
526	69	2	1	setxright(xright)
527				end!(initialize)
528				
529				begin (opcom)
530	1	1	1	initialize;
531	2	1	1	999:if myname=MAIN then
532	3	1	2	repeat
533	4	1	3	i:=0;
534	5	1	3	command:=
535	6	1	3	command:=
536	8	1	5	while not eoln do begin
537	9	1	5	read(ch);
538	12	1	5	if ch='a' then if ch='z' then ch:=chr(ord(ch)-32);
539	13	1	5	i:=i+1;
540	14	1	5	command[i]:=ch;
				buff[i]:=ch;

LINE	STMT LEVEL	NEST	SOURCE STATEMENT
541	15	1	5
542	17	1	3
543	18	1	4
544	19	1	5
545	20	1	5
547	21	1	5
548	22	1	5
549	24	1	6
550			
551	25	1	4
552	26	1	5
553	27	1	6
554	28	1	6
555	29	1	6
556	30	1	7
557	31	1	6
558	32	1	7
559	33	1	7
560	34	1	7
561	35	1	7
562	36	1	7
563			
564	37	1	6
565			
566	38	1	5
567	39	1	6
568	41	1	8
569	42	1	8
570	43	1	8
571	44	1	6
572	46	1	6
573			
574	47	1	3
575	48	1	3
576	49	1	2
577	50	1	3
578	51	1	3
579	52	1	3
580	53	1	3
581	54	1	4
582	55	1	3
583	56	1	4
584	57	1	4
585	58	1	4
586	59	1	4
587	60	1	4
588			
589	61	1	3
590			
591	62	1	1

ERRORS DETECTED: 0 WORDS
 FREE MEMORY: 7166

LINE STMT LEVEL NEST SOURCE STATEMENT

```

1      program regulator;
2
3      function adin(chan:integer):real; external;
4
5      procedure daout(chan:integer; value:real);
6          external;
7
8      type entryprocedures=(wait1, regputgetnode1,
9          getoutvalue1,getrealtime1,
10         datawrite1,getmyname1);
11
12     procedure entprocedure(entryprocedures): external;
13
14     const AD='AD'
15           DA='DA'
16           EXT='EXT'
17
18     type nametype=array[1..10] of char;
19
20     address=record
21         name:nametype;
22         number:integer
23     end;{address}
24
25     nodeType=(innode, PIDnode, outnode);
26
27     paramType=record
28         goalcomp:nametype;
29         intime,outtime,lasttime:integer;
30         case tag:nodeType of
31             innode:(insig,outputPlace:address;
32                 scale,filter:real);
33             PIDnode:(PIDin,PIDref,PIDout:address;
34                 k1,t1,d1,a1fa,Beta,limit:real);
35             outnode:(insig1,insig2,insig3,
36                 insig4,output:address;
37                 scal1,scal2,scal3,scal4,level:real)
38         end;{paramType}
39
40     statetype=record
41         filterstate,yold,ipart,dpert:real;
42         end;{statetype}
43
44     ddcnodeType=record
45         fwd,bwd:integer;
46         name:nametype;
47         priority:integer;
48         period,counter:integer;
49         outvalue:real;
50         parampart:paramType;
51         state:statetype
52     end;{ddcnodeType}
53
54
    
```

LINE	STMT	LEVEL	NEST	SOURCE STATEMENT
55				var node:ddcnodetype;
56				anyfound:boolean;
57				myname:nametype!
58				
59				procedure wait;
60				begin
61	1	2	1	entprocedure(wait1)
62				end;
63				
64				procedure regputgetnode(var node:ddcnodetype;
65				var anyfound:boolean);
66				begin
67	1	2	1	entprocedure(regputgetnode1)
68				end;
69				
70				procedure getoutvalue(var number:integer;
71				var outvalue:real);
72				begin
73	1	2	1	entprocedure(getoutvalue1)
74				end;
75				
76				procedure getrealtime(var rttime:integer);
77				begin
78	1	2	1	entprocedure(getrealtime1)
79				end;
80				
81				procedure datawrite(var comp, nodename:nametype;
82				var sendvalue:real);
83				begin
84	1	2	1	entprocedure(datawrite1)
85				end;
86				
87				procedure getmyname(var thisname:nametype);
88				begin
89	1	2	1	entprocedure(getmyname1)
90				end;
91				
92				function ulim(u,lolim,hilim:real):real;
93				begin
94	1	2	1	ulim:=u;
95	2	2	1	if u<lolim then ulim:= lolim;
96	4	2	1	if u>hilim then ulim:= hilim
97				end:(ulim);
98				
99				function nodevalue(var addr:address):real;
100				var value:real;
101				begin
102	1	2	1	if addr.name=AD then value:=adin(addr.number)
103	3	2	2	else getoutvalue(addr.number,value);
104	4	2	1	nodevalue:=value;
105	5	2	1	end:(nodevalue);
106				
107				procedure setouttime(var intime,outtime,lasttime,
108				period:integer);

LINE	STMT	LEVEL	NEST	SOURCE STATEMENT
109				var realtime:integer;
110				begin
111	1	2	1	getrealtime(realtime);
112	2	2	1	outtime:=realtime;
113	3	2	1	if (outtime - intime)=(period div 2)
114	4	2	2	then lasttime:=outtime;
115	5	2	1	end; (setouttime)
116				procedure inreg;
117				var value,ui:real;
118				begin
119				with node,parampart do begin
120	1	2	1	if insig.name()=EXT
121				then with node,state do begin
122	3	2	4	value:=scale*nodevalue(insig);
123	4	2	6	value:=(1.0-filter)*filterstate+filter*value;
124	6	2	6	filterstate:=value
125	7	2	6	end (then)
126	8	2	6	else value:=node.outvalue;
127				end (then)
128	9	2	4	ui:=ui*(value-1.0,1.0);
129				end (with)
130				end (inreg)
131				end (with)
132	11	2	3	if (goalcomp=myname) and (outplace.name=DA) then
133	12	2	4	begin
134	13	2	5	setouttime(intime,outtime,lasttime,node.period);
135	14	2	5	dout(outplace.number,ui)
136				end
137	15	2	4	else if (goalcomp=myname)
138				and (outplace.name=DA) then
139	16	2	5	begin
140	17	2	6	setouttime(intime,outtime,lasttime,node.period);
141	18	2	6	dout(outplace.name,value)
142				end
143				end (with)
144				end (inreg)
145	19	2	1	node.outvalue:=value
146				end (inreg)
147				end (with)
148				end (inreg)
149				end (with)
150				end (inreg)
151				end (with)
152	1	2	1	with node,parampart do begin
153	3	2	3	with node,state do begin
154	5	2	5	yr:=nodevalue(PIDref);
155	6	2	5	y:=nodevalue(PIDin);
156	7	2	5	e:=yr-y;
157	8	2	5	ipart:=ipart+alfake;
158	9	2	5	dpart:=beta*(yold-y);
159	10	2	5	u:=k*e+ipart+dpart;
160	11	2	5	u:=u*lim(u,-limit,limit);
161	12	2	5	yold:=y;
162	13	2	5	if (ti)>0.0 then ipart:=u-dpart-k*e

LINE	STMT	LEVEL	NEST	SOURCE STATEMENT
163				{anti-rreset-windup}
164				end; {with node.state}
165	15	2	3	if (goalcomp=myname)
166				and (PIDout.name=DA) then
167	16	2	4	daout(PIDout.number,u) else if
168				(goalcomp=myname)
169				and (PIDout.name=DA) then
170	18	2	5	datawrite(goalcomp,PIDout.name,u);
171	19	2	3	node.outvalue:=u
172				end {with node.parampart}
173				end;{PIDreg}
174				
175				
176				
177				Procedure outreg;
178				var out,out1:real;
179	1	2	1	begin
180	3	2	3	with node.parampart do begin
181	4	2	3	out:=level;
182	5	2	4	if scal1()>0.0 then
183	6	2	3	out:=out+scal1*nodevalue(1sig1);
184	7	2	4	if scal2()>0.0 then
185	8	2	3	out:=out+scal2*nodevalue(1sig2);
186	9	2	4	if scal3()>0.0 then
187	10	2	3	out:=out+scal3*nodevalue(1sig3);
188	11	2	4	if scal4()>0.0 then
189	12	2	3	out:=out+scal4*nodevalue(1sig4);
190	13	2	3	out1:=uliw(out,-1,0,1,0);
191	14	2	3	node.outvalue:=out;
192				if (goalcomp=myname)
193	15	2	4	and (output.name=DA)
194				then daout(output.number,output1) else if
195				(goalcomp=myname)
196	17	2	5	and (output.name=DA)
197	18	2	3	then datawrite(goalcomp,output.name,output);
198				end {with}
199				end;{outreg}
200				end;{regul}
201	1	1	1	begin {regul}
202	2	1	1	getmyname(myname);
203	3	1	1	repeat
204	4	1	2	wait;
205	5	1	2	regputgetnode(node,anyfound);
206	7	1	4	while anyfound do begin
207	8	1	5	case node.parampart.tag of
208	9	1	5	innode: inreg;
209	10	1	5	PIDnode: PIDreg;
210				outnode: outreg
211	11	1	4	end;{case}
212				regputgetnode(node,anyfound)
213				end {while}
214	12	1	1	until false;
				end {regul};

ERRORS DETECTED: 0
 FREE MEMORY: 9000 WORDS