Universitat Autònoma de Barcelona
Dpt. Enginyeria Química
08193 Bellaterra, Barcelona,
Spain

MELISSA

Memorandum of Understanding

ECT/FG/MMM/97.012

Contract Number: ESTEC/CONTRACT11549/95/NL/FG

TECHNICAL NOTE: 37.9

General Purpose Station.

Software upgrade 97

Version: 1

Issue: 0

PONS P.; ALBIOL, J.; GODIA, F.

APRIL 1998

Document Change Log

Version	Issue	Date	Observations
Draft	0	16/04/98	Preliminary Version
1	0	1/06/98	Original Version

TABLE OF CONTENTS

1.- INTRODUCTION. **5**

2 .- IMPLEMENTATION OF CHANGES IN THE MATHEMATICAL MODEL USED FOR THE CONTROL OF COMPARTMENT IV **7**

2.1 MODIFICATION OF THE METHODOLOGY FOR CALCULATION OF LIGHT INTENSITY (FR). 8
2.2 SUPPRESSION OF THE GAP BETWEEN PREDICTED AND MEASURED PRODUCTIVITY VALUES. 9

3.- INCORPORATION OF A SIMULATOR IN THE GPS. **10**

3.1 .- MODIFICATIONS IN THE SOURCE CODE TO INCLUDE THE SIMULATOR. 12
3.2 .- MODIFICATIONS IMPLEMENTED IN THE SIMULATOR SOURCE CODE. 13
3.3 .- CONFIGURATION OF THE SIMULATOR. 14

4 .- COMMUNICATION WITH THE SYSTEM STATION **15**

5.-EVALUATION OF THE MELISSA CONTROL SYSTEM SET-UP **16**

5.1 GENERAL DESCRIPTION OF THE PRESENT CONFIGURATION. 16
5.2 LIMITATIONS OF THE SYSTEM 17
Operating system. 17
Interchanging data between modules. 19
Interrelation between modules. 20
Dependence of one vendor. 20
A new orientation. 21
Software design. 22
A more defined proposal. 22
How the evolution affects FLS software. 23
Step by step upgrade 24

6.- REFERENCES **25**

7.- SOURCE CODE. **26**

HEADERS 26
Alarms.h 26
CtrlLiqu.h 27
CtrlNitr.h 28
CtrlRhod.h 29
CtrlSpir.h 30
Help.h 32
Melfct.h 33
Melissa.h 34
Pargene.h 37
Parmodel.h 38
Results.h 39
Screen.h 40

Simfile.h 41
UserDef.h 42
Vars.h 56
C-FILES. 57
Alarms.c 57
CtrlSpir.c 60
CtrlNitr.c 73
CtrlLiqu.c 75
CtrlRhod.c 77
GpsFile.c 78
Graph.c 81
Help.c 87
MelFct.c 89
Melissa.c 91
ModSpiru.c 105
Results.c 116
Screen.c 133
Vars.c 154
Qbs.c 164

1.- INTRODUCTION.

The successful and reliable operation of a compartmentalised system such as the MELISSA loop, requires the development of an advanced control system. Its main task being the co-ordination of the behaviour of each compartment to assure the crew survival. The General Purpose Station (GPS) constitutes the nucleus of the loop and it has been conceived in such a way so as to allow the step by step development and implementation of such a control system.

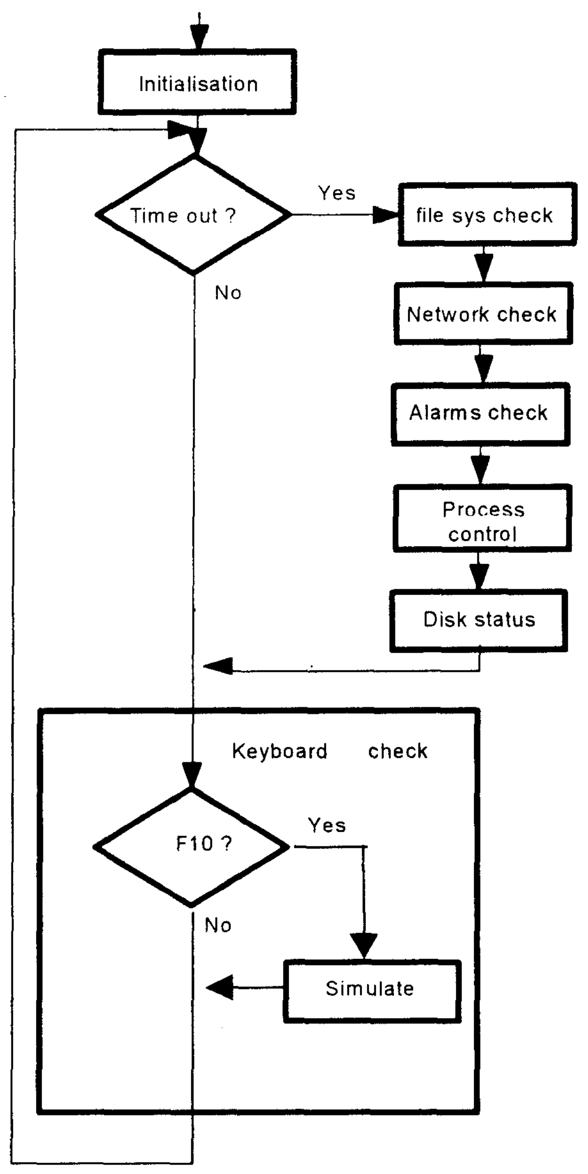


Figure 1

The recent developments in the MELISSA Pilot Plant, have introduced a number of modifications, including the installation of a new bioreactor, equipped with a new type of controllers. At present time, three bioreactors are installed in the pilot plant, corresponding to compartments II, III and IV. This technical note (TN) describes the different changes introduced in the GPS following its evolution, and it up-dates the previous information on the GPS provided in TN 25.5 (Pons *et al* 1998).

The contents of this technical note is divided in three parts. First, the changes introduced after a revision of the mathematical model describing the relationship between the growth of *Spirulina* cells and the illumination conditions of the air-lift bioreactor are described. Secondly, the incorporation of a simulator to the GPS is reported. Such a simulator is an ADERSA implementation in C language of the one previously developed in FORTRAN and described in TN 19.3 (Cornet et al. 1993). The simulator allows to simulate the biomass composition of the *Spirulina* cells (in terms of the percentage of different compounds) as a function of the operational conditions. Finally, the upgrade of the GPS in order to incorporate the new group of controllers corresponding to the bioreactor of compartment IV in the Pilot Plant is discussed.

2 .- IMPLEMENTATION OF CHANGES IN THE MATHEMATICAL MODEL USED FOR THE CONTROL OF COMPARTMENT IV.

As it was described previously in TN 25.5 (Pons *et al.* 1995, section 1.8), the main loop of the GPS software, for the control of the different compartments of the MELISSA Pilot Plant was designed according to the structure presented in figure 1 and figure 2.

The main goal of this structure, with the proper source code organisation at module level, is to enable the modification of any part of the control system without creating any disturbance in the rest of the software. This structure allows then to develop the model of each individual compartment. With the progress of the work performed in the MELISSA project, the complete content and performance of this GPS structure is gradually updated.

Currently, the most developed of the different modules shown in figure 2 is the one corresponding to compartment IV, referred as "*Spirulina control*". The corresponding software module that contains the control routines developed by ADERSA for this compartment are tagged as "Ctrl Spir". As a consequence of the work developed recently in compartment IV, two different modifications were introduced in the mathematical model and its use for control purpose, and consequently they have been incorporated into the global control system program in the GPS, as will be described below.

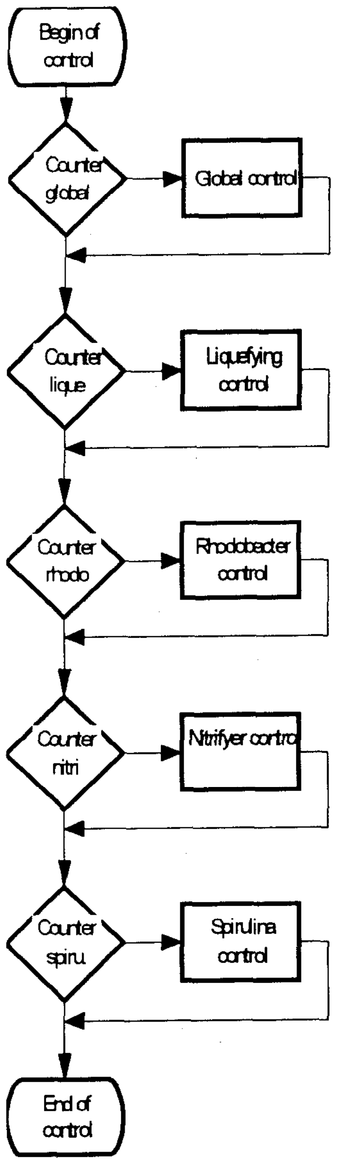


Figure 2.

2.1 Modification of the methodology for calculation of light intensity (Fr).

The predictive control of compartment IV is based in the mathematical model that links the illumination performed in the bioreactor and cell growth. The controlled variable is the light intensity, measured as the amount of light energy supplied to the bioreactor, per surface unit (W/m^2), and referred to as Fr. Light energy is supplied by halogen lamps located around the reactor. The control system can automatically modify the light intensity by changing the voltage applied to the lamps. However, the Fr value can not be measured directly. For this reason, the set-up of the compartment IV

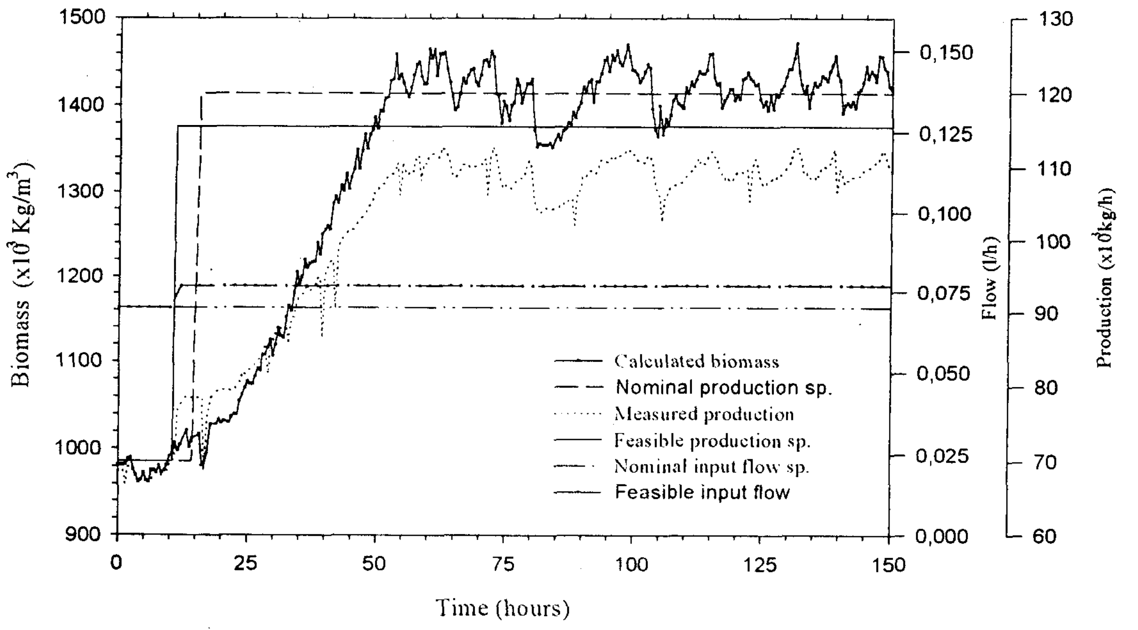


Figure 3 : Example of the profiles obtained in a continuous culture run of *Spirulina platensis* after the

bioreactor included a sensor in the middle of the air-lift bioreactor. By using a light distribution model (which takes into account the reactor geometry, cell absorption and scattering), the Fr could be estimated from the value of the light energy measured in the centre of the bioreactor.

This system is highly complex, and it was observed that its prediction of Fr was not accurate for certain situations, such as low light values in the centre of the bioreactor. This fact, together with other arguments of practical type, such as the fragility of the sensors, suggested to change the system used for the calculation of Fr.

The system currently used is based on a calibration plot prepared before the real operation of the reactor. The calibration function establishes the relationship between the voltage applied to the lamps and the light intensity, Fr, and it is easily measured when the bioreactor is empty, measuring directly the light in the centre of the bioreactor and converting the measure to the value at the bioreactor surface. Being the bioreactor empty, it is not necessary to take into account cell absorption and scattering, which is the main fact introducing complexity in Fr calculation. The measures are performed at different bioreactor heights and then averaged, as has been reported previously (Vernerey *et al.* 1997). The procedure is repeated for different values of the range of the operational lamp voltages.

This change was implemented in the GPS, and used for the control of a continuous culture run of compartment IV. The obtained results are discussed in TN37.110 (Vernerey *et al.* 1997). The general trend observed in the system was correct, but it was observed that the biomass production was always a certain percentage lower than predicted by the control system. This fact can be observed in figure 3, and suggested a new modification of the control software, described in the next point.

2.2 Suppression of the gap between predicted and measured productivity values.

After the successful implementation of the first modification, a second step towards improvement was undertaken. As mentioned earlier, during a experimental run and once the steady state was reached, it was observed that between the productivity set-point and the measured value it remained a small gap that did not decrease with time. Instead, the measured variable remained under the set-point value. To improve this behaviour ADERSA, modified the control algorithm in order to eliminate this small discrepancy.

The corresponding modification was done directly in the source code of the control module by ADERSA personnel and delivered to the Pilot Plant for insertion into the GPS code files. As a result of the modified control laws, the control was improved, as can be seen in figure 4, and the measured production presents a very small oscillation around the set-point.

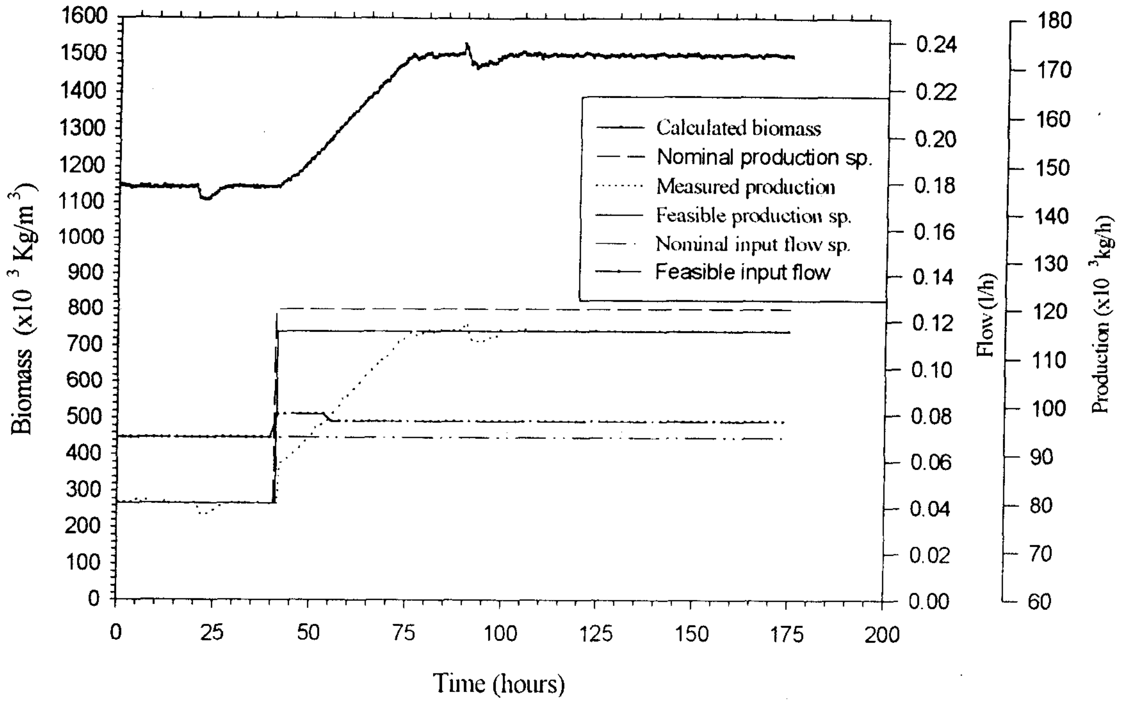


Figure 4 : In this figure the more accurate results of the control laws can be observed.

3.- INCORPORATION OF A SIMULATOR IN THE GPS.

As mentioned in the introduction, a simulator was incorporated to the GPS. This software allows the user to predict and evaluate the consequences of the changes in the operation of the compartment on the biomass quality. That is the values of the content in the different groups of macromolecules as shown in table 1.

The simulator is based on the first principles model built by J.F. Cornet (LGCB Clermont-Ferrand) for simulation of cultures of the cyanobacterium *Spirulina Platensis* cultivated in cylindrical and radial illuminated photobioreactors. The set of routines of the simulator included in the GPS are an adaptation, programmed in C language, of the simulator Photosim, described in TN 19.3.

The simulator inputs are:

- the incident radiant energy flux in W/m^2,
- the dilution rate in 1/h

- the concentrations in the incoming flow of the nine compounds in kg/m^3 (see table 1)
- the initial concentrations in the reactor of the nine considered compounds of the biomass that are taken into consideration in this simulator (table 1).
- As outputs, the simulator generates the following data :
    - concentrations in the reactor of the nine considered compounds
    - composition of the total biomass expressed in mass fraction (dimensionless).
    - and global formula of the total biomass expressed with the chemical elements C, H, O, N and P.

The output data is stored in plain text files in the directory named "\gpssim", and those files are graphically represented into the GPS screens. The generated files have the same format that the existing output files, in order to facilitate its set-up.

Compounds.
Total biomass
Active biomass
Chlorophyll
Phycocyanin
Protein
Nitrate
Sulfate
Vegetative biomass
Exopolysaccharide

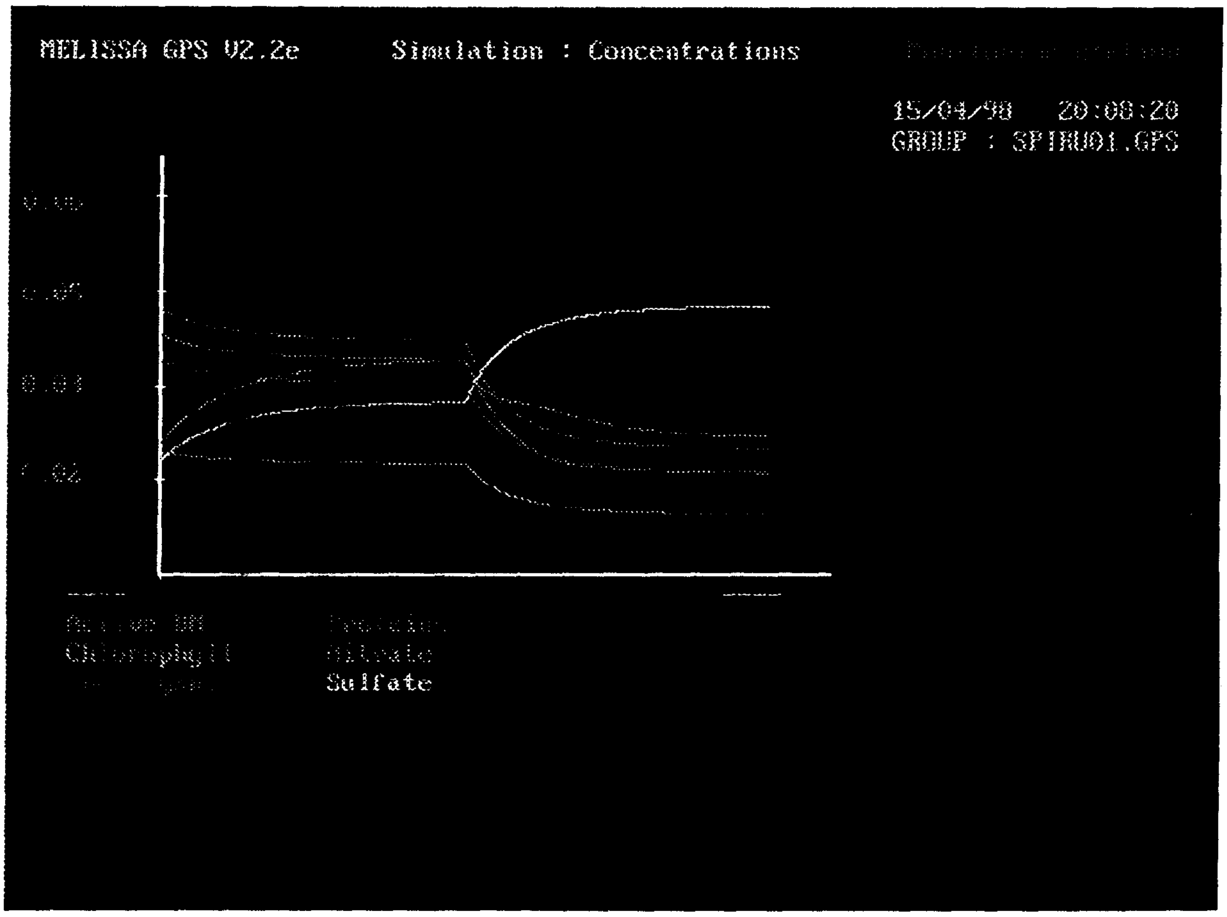
Table 1 : List of the components taken into account in the simulator

3.1 .- Modifications in the source code to include the simulator.

The inclusion of a new screen, to show graphically the results of the simulation, requires the addition of an instance of one structure and its correct filling. Therefore, the inclusion of the simulator in the GPS, required changes in the keyboard control routine, the initialisation routine – to fill the screen structure –, and the modification of the simulator code to accommodate it to the compiler and environment. The source code is listed in the last part of this document, for documentation purposes,.

The source code has been included in the module named modspiru.c. The simulation routines are called from the keyboard manager routine when control and F10 are pressed. Graphical representation of the data is obtained with the press of F9 (Concentration), Ctrl-F9 (Global formula) and Alt-F9 (Biomass fraction).

In figure 5, an example of a screen showing the results of a simulation is presented.



```
MELISSA GPS V2.2e       Simulation : Concentrations

                                                    15/04/98    20:08:20
                                                    GROUP : SPIRU01.GPS
```

Figure 5 : Example of screen showing simulation results.

3.2 .- Modifications implemented in the simulator source code.

The operating system used currently in the GPS, MS-DOS, imposed some limitations on the direct use for the simulator. In particular the memory is divided in segments of 64 kbytes, while some of the data matrix are greater that 64 kbytes. To overcome this problem, instead of storing the data matrixes in memory, they were stored in a disk file. This modification forced also to change the pass of parameters to the simulator routines. For similar reasons the output of the simulations, was done also to a disk file. The use of a disk file as a solution, does not impose a strong performance penalty, provided that a disk cache is concurrently used.

Once the modifications were implemented, several simulation tests were done to compare the results with the original simulator.

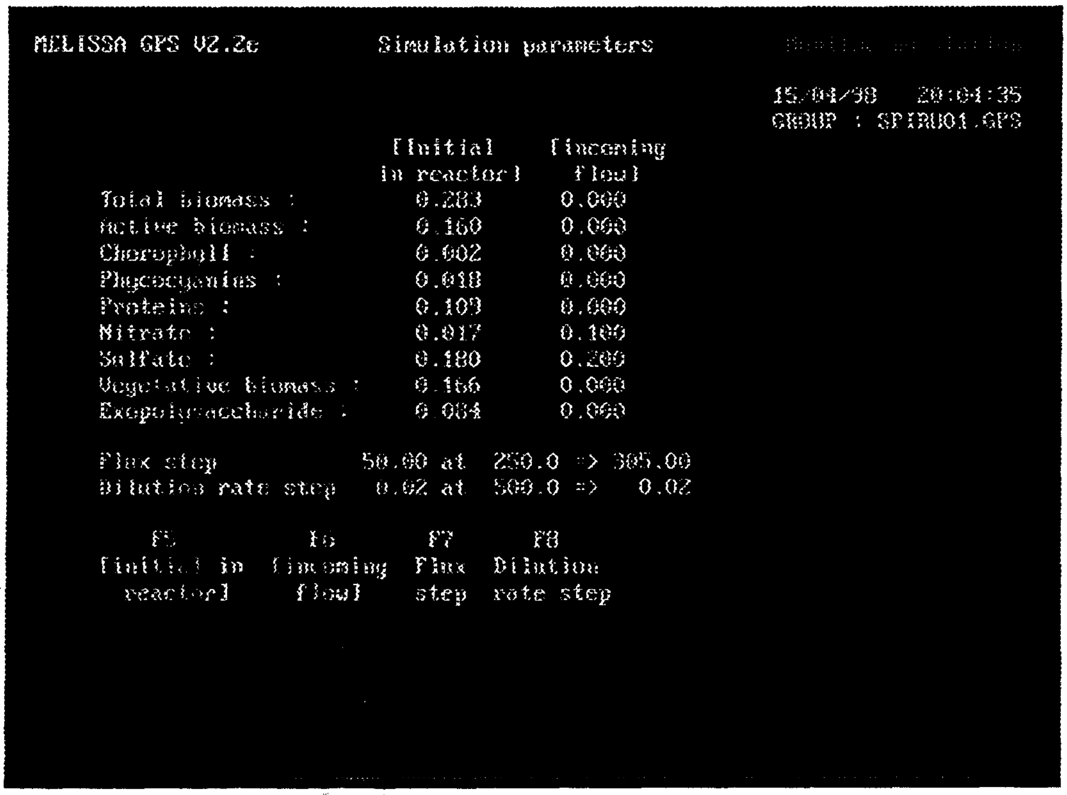
3.3 .- Configuration of the simulator.

A combination of keys (Ctrl + F10) has been reserved to access the configuration screens. The configuration screen layout is divided in three main areas:

- the initial reactor data column
- the incoming flow data column
- the flux step and dilution rate row area.

These areas present the information that the simulator uses as initial states.

If the user press some of the function keys that appear under the dilution rate row, the screen enters in a different mode, the update mode. This mode allows the user to configure the initial state of the simulator. Internally, these new state stores the values in configuration files located in the directory \gpssim.



```
MELISSA GPS V2.2c          Simulation parameters

                                                          15/04/98   20:04:35
                                                          GROUP : SPIRU01.GPS
                              [Initial    [incoming
                              in reactor]   flow]
     Total biomass :            0.283       0.000
     Active biomass :           0.160       0.000
     Chorophyll :               0.002       0.000
     Phycocyanins :             0.018       0.000
     Proteins :                 0.103       0.000
     Nitrate :                  0.017       0.100
     Sulfate :                  0.180       0.200
     Vegetative biomass :       0.166       0.000
     Exopolysaccharide :        0.084       0.000

     Flux step              50.00 at  250.0 => 305.00
     Dilution rate step      0.02 at  500.0 =>   0.02

         F5          F6        F7     F8
     [initial in  [incoming  Flux   Dilution
       reactor]     flow]    step   rate step
```

Figure 6 : Example of the configuration screen of the simulator. For this tests, a monitoring version of the GPS was used so as not to disturb the control of the plant.

4 .- COMMUNICATION WITH THE SYSTEM STATION.

The system station is a computer, or set of computers, that execute a commercial program. This station runs a TOPTOOLS program that receives the data from the controllers, and its logical location is between the controllers and the rest of computers. One of its tasks is to transfer the obtained data from the MICON or ASCON controllers to the GPS. This program performs some other important tasks such as data collection, allowing a direct access to some memory positions of the controllers, and the display of historical data.

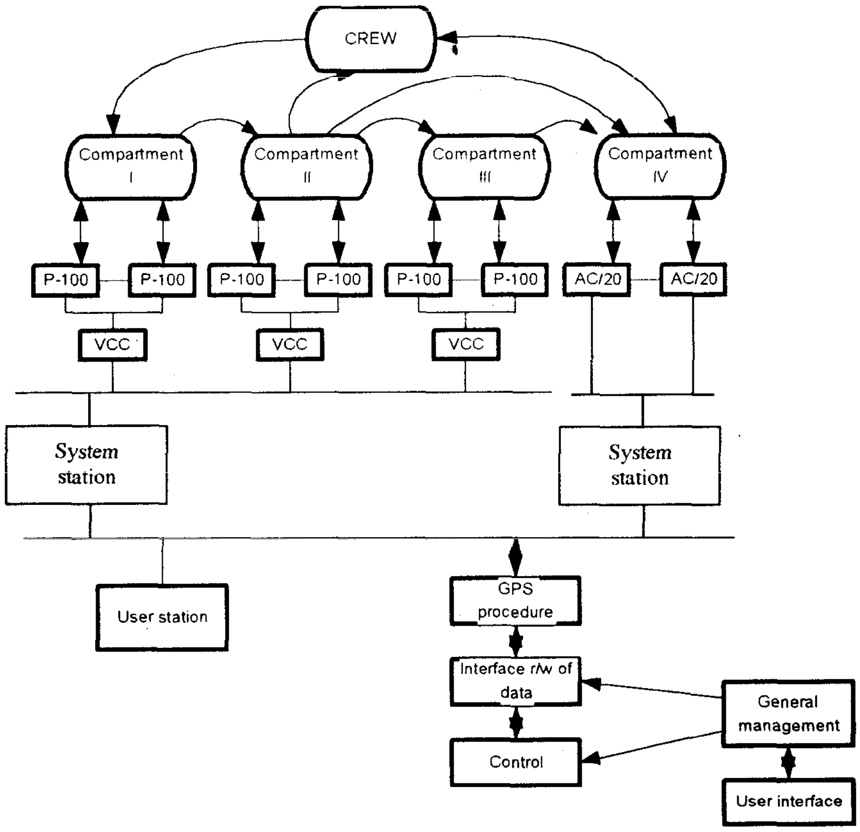


Figure 7 : General scheme of the plant.

Due to the incorporation in the Pilot Plant of a new 75 litres bioreactor, new ASCON controllers were purchased. In order to make available, the data collected by this controllers, to the software already installed in the Pilot Plant, an upgrade of this software was necessary. As a first step in the upgrade of the system the original INDUSTAR Toptools version 3.4 was upgraded to V 3.6. After installation, the software is able to communicate with the ASCON controllers and store the data in the same way as it was done for the MICON controllers.

5.-EVALUATION OF THE MELISSA CONTROL SYSTEM SET-UP

This section describes the actual configuration of the MELISSA Pilot Plant control system, pointing out its limitations. At the same time, possible solutions are suggested to be implemented in the future expansion. The evaluation will be focused in several topics such as interoperability, expansion of the GPS, and dependence of only one manufacturer. A second part of this chapter will describe some important guidelines in order to obtain of a control system that can fulfil the Melissa requirements.

5.1 General description of the present configuration.

The Melissa control system has at its lower level (level 0 in the control strategy) a set of programmable controllers, that are able to maintain the value of the controlled variables at its set point values, in an autonomous way. Those controllers are connected to computer stations either directly or through vertical communication controllers. The low level controllers can run simple control strategies that are programmed using proprietary software. Two different models of controllers are used at present time.

The previous models used were Sensycon P-100 MICON controllers while the newest models are ASCON AC-20. The ASCON AC20 communicate with the computers using the standard Modbus protocol. The older Sensycon P-100 MICON controllers communicate with the computers using a proprietary protocol.

There are two computers connected to the controllers both of PC type. One of them has as a CPU an Intel 80486 and the other one an Intel 80386 both from the manufacturer COMPAQ. The communication between the computers and the controllers is done using an expansion card that provides an RS-485 interface.

The computers connected to the controllers run a dedicated software for this task which allows either the manipulation of the parameters and set points of the controllers, as well as the storage and graphical representation of the data collected and alarm management. The computers are interconnected using an Ethernet type local area network, using an IPX/SPX Novell protocol. The software run by the computers is the INDUSTAR TOPTOOLS with different modules such as a control and command module, a data storage module, a configuration module, a graphical data display module or a global purpose station module (GPS). The original INDUSTAR version 3.4 has been recently upgraded to V 3.6. This software runs under the MS-DOS platform V-6.0.

The GPS is in fact a library of functions in C language that allow access to the controllers variables, using another computer station connected to the same network. The last version available of this library has to be linked with the linker provided in the version 6.0 of Microsoft C/C++. This library allows to develop a more advanced control system by means of building a specialised control program. In the present configuration this control program makes use of advanced mathematical knowledge models, for the control strategy and takes command of the control actions of the low level controllers.

5.2 Limitations of the system

Due to the time elapsed since the first system was installed and the natural evolution of computer software in the last years, the MELISSA control system presents several limitations either intrinsically or in comparison with the state of the art control software used currently day personal computers. In the following points, the most important of those limitations will be mentioned and possible solutions will be proposed at the end of the chapter.

Operating system

As mentioned before, the computer software used in the control system was compiled to be executed in personal computers under the MS-DOS operating system. This includes either TOPTOOLS and the GPS library subroutines. This operating system was designed to be compatible with the Intel 8086 microprocessors, and consequently suffers several limitations that have been overcame by modern operating systems (OS) like Windows-95, Windows-NT or UNIX to mention just the more common. The more important limitations from the point of view of the MELISSA control system are the non availability of multitasking or multithreading, memory address using segmentation (which in practical terms limits the size of one data item like a matrix to a segment of 64kb), difficult debugging due to the few facilities offered by the old compilers (Microsoft C 5.8, is not able to perform an acceptable type-checking), resulting in longer development times, and the impossibility to use newer hardware elements that are being sold only with drivers for the newer OS. All of those elements do not allow to use all the power available in the new families of computers having Pentium of Pentium II microprocessors.

The limitations due to the non availability of the multithreading or multitasking capabilities will be more and more evident as soon as the control model is going to become more complex with the incorporation of control routines for several compartments at the same time, due to the fact that it is not possible to have the control routines for each compartment running in an efficiently independent way without the intervention of the OS.

The new OS will allow to have in one program the execution of different threads in a time independent way with minor interference. In case that this minor interference is still important, the new operative systems allow to use computers with more than one CPU or even execute different parts of the control strategy in a distributed way. That is they allow for the execution of parts of the code in different computers in a transparent way. Development of software would be even more compartmentalised and easy due to the fact that each group contributing to the software can supply its own dynamic link library (dll) provided they follow a standard convention to allow for execution of its subroutines from the main program.

In consequence, from the interoperability point of view, this system only allows the use of MS-DOS systems in a LAN for accessing the plant. This library is not able to intercommunicate several MS-DOS processes, resulting in a very limited system for nowadays standards.

The latest GPS library available in the lab was compiled for Microsoft C V6.0. As this version is not available in the lab, Microsoft 7.0 was used. On one side this has resulted in minor improvements. On the other side the vendor only guarantees its libraries if they are used with the compiler they were designed for. In consequence the guarantee of the vendor is lost. Anyway this library was compiled with a compiler designed for 16 bit computers and therefore it appears not possible to use it with the new 32 bit compilers used for the more advanced OS like Win-95, NT or UNIX.

An interesting ideal solution to all this would be a system that could allow the use of several operating systems intercommunicated with an standard communication protocol so that it could be used a UNIX system, a Windows NT/95 system or any other state of the art one.

GPS Expansion.

The control of the plant is organised in a hierarchical way, so that the controllers cover the lowest control level and higher levels are located in the General Purpose Station software. This GPS contains the mathematical models that are used to predict the evolution of the cultures.

Due to the progressive advance of the Melissa loop, the number of reactors to be controlled by the GPS is increasing as well as the number of mathematical models and computing to be done. The problem of calculation requirements is resolved using more powerful computers – Pentium II instead of 386 processors –, but the problem of the program size is still important.

The source code is organised in modules, where each module has a clearly defined responsibilities. However there can be some interferences among the different modules, specially if the time to execute each module is taken into account.

The problem of the size of the code is that this size can not be changed because all the features of the GPS are supposed to be important. The problem that therefore appears is that the compiler can not manage modules too large, global variables that remain in the code due to the initial design, and some matrixes that are used in the simulator of the *Spirulina* cultures as inputs and outputs , are too large.

Exchanging data between modules.

As is explained in TN 18.3, the structure of the original GPS that is used to pass data among different compartments is the global space section. This structure will become more problematic with the expansion of the system. The problem that now is going to appear is that the size dedicated to global variables by the compiler – in the larger memory model – is 64 Kilobytes, and all the structures that represent the variables have to be fitted in this area. The result is that the entire program will have to be modified to make possible the inclusion of three new compartments. From this point of view the possibility to build a new GPS from zero, using state of the art tools, appears as the most promising in the long term.

Interrelation between modules.

In the present day GPS, there is just one execution thread. This means that just one task can be performed at a time. Currently this is not an important problem because the system is composed by only one compartment control module, but in the near future there will be more than one compartment to control and possibly some simulators. The inclusion of various simulators will cause to spend too much time in processes that are not oriented to control the plant and can affect the control tasks. Today the problem is no serious because the data reading procedure takes only some seconds, and the simulator is turned on by a key pressed in the keyboard. However when the data will have to be read from different compartments, filtered and some other tasks performed, there is a high probability that the time used will be longer than the sampling period. Today, execution of the simulator in some computers takes over two minutes, which is too much time because the sampling period is one minute.

Dependence of one vendor.

One of the most important goals in the market is compatibility, that is to say, the use of one product does not force to have other related products of the same vendor. One of the advantages of this strategy is generally the high quality of the products, and as a result of this compatibility the user can change one piece of software for another without important economical costs. On the other hand, the non-compatible products force the user to buy all the equipment, software and technical advice to the same vendor. A change of product in this case is more expensive and the vendor is safer knowing the important expenses that the user must commit to modify the system.

This last case is today the case of Melissa. The software used is provided by FLS, and the controllers by ASTARE. Due to some incompatibilities, some changes have had to be performed in the software to allow the AC-20 controllers to work with TOPTOOLS. These situation implies that a potential change in the Plant system can imply the change of the software itself, what implies the programming of a new GPS, and – depending on the new software –, the MICON controllers.

A new orientation.

Having in mind that the main goal is to obtain an expandable system, in this section a set of points will be presented so that to guarantee the features of expansion, low cost of maintenance and reusability of components.

□ Operating system:

It is important to use a modern operating system that allow the programmer to exploit the whole computer used. The main features supported by the operating system must be:

□ Support of multithreading and multiprocess techniques.

□ Support of Tpc/Ip network protocol.

□ Facility for the construction of distributed applications.

□ A friendly GUI (Graphic User Interface).

□ Compiler :

This is another important point. It must be a C and C++ compiler fully compatible with the compiler used by ADERSA. It must be provided by a vendor consolidated in the market or very extended. It can be interesting the use of a compiler that allows the programmer to use a mixture of languages – one for the graphical interface, one for the control routines, etc. –. For example, coding the control routines in Visual C++ and the user interface in Visual Basic or Delphi, the development time is importantly reduced, and also forces a strong modularity. The development lapse is reduced due to the simplification of interface design that some languages offer (interface-oriented languages), as far as it also allows to take advantage of the powerful calculation facilities of other languages (as C, C++ or Fortran). From the other hand, a kernel implemented in real portable C, independent of the interface can help further evolutions.

□ Computers used.

The computers should Pentium and Pentium II from Compaq. It is going to be necessary the acquisition of, at least, one new computer because during plant operation it must be possible to develop the software without having the risk of disturbing the plant operation (for example by having to reboot a computer).

Software design.

The strategy followed in software construction should follow a design of a logical distributed structure. The logical distribution of the components of the GPS does not necessarily imply physical distribution of components through different computers around the LAN. In principle the idea is to develop the new GPS but having in mind that in the future, it is possible that the control for each compartment can be so complex that a entire computer is needed by compartment, or by task (simulation for example). With this concept in mind one solution could be the implementation of the control of each compartment in different processes and a higher control that communicate and co-ordinate these processes. While the control remains with a moderate level of complexity, all the compartment processes can be in the same computer, but when the complexity of the models was large enough, one compartment process could be moved to another computer without affecting the rest of the control system and without having to perform any change in the software.

One key point in this solution concept is the dynamic linking. That is to say, it should be possible to encapsulate the control routines inside a dynamic link library, the simulator inside another DLL, and any other component module subject to be modified confined inside another DLL. This solutions provides the facility of changing a version of the control laws just with the change of one file, saving time, avoiding mistakes and encapsulating responsibilities.

Another point subject to be changed, is the concept of the GPS like one block. It should be better to distribute the different control process, and related tasks in different processes. This processes can be distributed through the LAN or not, depending on the workload of each task. At the same time, this separation of tasks through different computers improves the fault tolerance of the system.

A more defined proposal.

In the last pages the general requirements of a computer system have been described. In this paragraph a more defined proposal is presented. Based on the knowledge of the system requirements it is possible to choose a group of operating systems that can be candidates to be used in the plant, a set of compilers and a set of languages.

One of the best operating systems that are available for PC is Linux. It is stable, there is no purchase cost, there are many programming tools of public domain, it works gratefully on the network and is very extended. The problem is that there is not a firm behind it, and nobody is responsible for its development or capabilities. This appears to be an important point. The next operating system of the list is Windows NT. NT is not as stable as Linux, but the continuity of the operating system and the available tools for it is guaranteed by Microsoft. NT present the advantages of easy multithread programming which allows the development of powerful applications with relatively low knowledge of the system. From this point, the rest of operating systems that could be interesting are commercial UNIX systems that are not so extended as the two mentioned above and not so well tested. About the tools, it is important to have in mind that in NT it is better to use Visual C++ 5.0 – or newer – and Delphi 3.0, and for UNIX systems the set of tools of GNU work very well.

How the evolution affects FLS software.

Changing the GPS and bringing forward the new concept of the system affects some other parts of the plant. The part that is directly affected is the software provided by FLS because is the main part of the currentl system that has to be changed. The reasons for its substitution are due to its proprietary communication protocol and the complexity of its problems. The first proposed solution is to take LabView (TM), with a much lower cost and allowing a complete programming of the protocol to use in the network, local control, data storing and it is very extended and many drivers are available for Windows NT. A second option is to purchase the new version developed by FLS. It is a system designed to be used in a chemical plant under Windows-NT. At first sight it appears a more expensive option, and due to its novelty it is probably less stable. Moreover the distant location of the supplier originates delays and increases the service cost.

As a final conclusion it must be assumed that a important change must be done to be able to control the plant when, in a close future, the Melissa loop will be completed.

This change must affect the initial conception of the software because the actual technologies are not oriented to serial task but to concurrent tasks, and any serial conception of the will create bottlenecks.

Step by step upgrade

The necessary upgrade of the pilot plant control system has to be done without disturbing the normal operation of the plant. Development and test of the different control laws and data collection for mathematical modelling have to continue independently of the GPS upgrade. On the other hand it has already pointed out that the GPS is limited by the memory available in MS-DOS and therefore that the inclusion of more executable code will be difficult before changing the operative system. It is therefore necessary a preliminary solution that allow the continuation of experimentation. The solutions proposed in this preliminary case will be discarded as soon as the new system is installed.

As a first step it will be determined if the inclusion of the control code corresponding to one new compartment can be included in the GPS. If the new control subroutine is not yet available, the control routine of *Spirulina* reactor can be replicated and included as if it were the new control routine. This way it can be determined if the compiler will support an increase of approximately equal amount of code. If a successful result is obtained, the control of the *Rhodospirilum rubrum* compartment will be included into the GPS as soon as it is available.

In case the new amount of code can not be included in the GPS, the code will have to be distributed among two systems. That means to replicate the GPS in two computers. On one computer using the *Spirulina* control system, and on the other one the GPS with the *Rhodospirilum Rubrum* control routines instead of the *Spirulina* ones. Using this solution each GPS should access and control only one reactor, minimising the interference among the two controller routines, and allowing to work independently in its maintenance.

6.- REFERENCES

PONS P.; ALBIOL J.; GODIA F.;1996. General Purpose Station. Main Program Update . ESTEC/CONTRACT 11549/95/NL/FG Technical Note. 25.5.

CORNET J..F., DUSSAP C.G.; GROS J.B. 1993 Applications to Simulation and Control of the *Spirulina* Compartment of the MELISSA Artificial Ecosystem. ESTEC/CONTRACT PRF 130820 Technical Note. 19.3.

VERNEREY A.; ALBIOL J.; GODIA F.; 1997 Control Laws of Photosynthetic Compartment. Biomass Light Studies. ESTEC/CONTRACT 11549/95/NL/FG Technical Note. 37.110

7.- SOURCE CODE.

Headers

Alarms.h

```
#ifndef __alarms_h
#define __alarms_h

/*********************************************************************

        NAME            ALARMS.H

        AUTHOR          Pedro L. Pons

        DESCRIPTION
                ALARM management listing file

        UPDATES
                25-05-96

*********************************************************************/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"
#include "ctrlspir.h"

/*-------------------------------------
        variables declarations
-------------------------------------*/

VARS    rst_alarm1;     /* Reset alarm P100 1 */
VARS    rst_alarm2;     /* Reset alarm P100 2 */
VARS    rst_alarm3;     /* Reset alarm P100 3 */
VARS    rst_alarm4;     /* Reset alarm P100 4 */

VARS    rst_main_alarm;     /* Reset main alarm */
VARS    rst_micon1_alarm;     /* Reset micon1 alarm */
VARS    rst_pressure;
/*VARS    active_lamp;     */
VARS    alarm_micon1;
VARS    alarm_micon1_value;

extern int activated_alarms;

#endif
```

CtrlLiqu.h

```
#ifndef __ctrlliqu_def
#define __crtlliqu_def
/**********************************************************************

NAME            CTRLLIQU.H

AUTHOR          Pedro Pons

DESCRIPTION     Headers for the lique controler.

UPDATES
      27-05-96

**********************************************************************/

#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"
#include "results.h"

/*--------------------------------------
      variables declarations
--------------------------------------*/

#endif
```

CtrlNitr.h

```
#ifndef __ctrlliqu_def
#define __crtlliqu_def
/*****************************************************************

NAME            CTRLNITR.H

AUTHOR          Pedro Pons

DESCRIPTION     Headers for the nitrogen controler.

UPDATES
      27-05-96

*****************************************************************/

#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"
#include "results.h"


/*-------------------------------------
      variables declarations
-------------------------------------*/

VARS NitrDo;
VARS NitrpH;
VARS NitrTemp;
VARS NitrPressure;
VARS NitrAmCon;
VARS NitrNitrCon;

int next_pfc_nt;
int first_time_nitrogen;
#endif
```

CtrlRhod.h

```
#ifndef __ctrlliqu_def
#define __crtlliqu_def
/*******************************************************************

NAME            CTRLROD.H

AUTHOR          Pedro Pons

DESCRIPTION     Headers for the rhodo controler.

UPDATES
      27-05-96

*******************************************************************/

#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"
#include "results.h"



/*---------------------------------------
      variables declarations
---------------------------------------*/

#endif
```

CtrlSpir.h

```
#ifndef __ctrlspir_def
#define __crtlspir_def
/*******************************************************************

NAME            CTRLSPIR.H

AUTHOR          Pedro Pons

DESCRIPTION     Headers and declarations for the spiruline controler.

UPDATES
      27-05-96

*******************************************************************/

#include <malloc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "userdef.h"
#include "melissa.h"
#include "results.h"

int my_interrupt();

/*---------------------------------------
      variables declarations
---------------------------------------*/

VARS    cxa;              /* biomass concentration              */
VARS    nitrate;          /* nitrate concentration              */
VARS    cal_nitrate;      /* nitrate calibration switch         */

VARS    Eb;               /* light intensity in the reactor     */
VARS    Fr;               /* incident flux                      */
VARS    temperature;      /* temperature in the reactor         */
VARS    pH;               /* pH of culture                      */
VARS    act_pompe;        /* dilution pump action               */
VARS    act_lamp;         /* lamp action (dimensionless)        */
VARS    cal_pump;         /* calibration of pump (l/h)          */

/*******************     variables ADERSA   ******************/

VARS    cons_prod_nom;    /* nominal production setpoint        */
VARS    cons_prod_real;   /* feasible production setpoint       */
VARS    qe_nom;           /* nominal flow setpoint              */
VARS    qe_real;          /* feasible flow setpoint             */
VARS    production;       /* measured production                */
VARS    prod_mod;         /* model production                   */

double  sm_sup;       /* model ouput (internal model of the supervisor)
*/
double  cons_sup;     /* ouput of the supervisor
*/

int     next_pfc_sp;         /* next execution of PFC for spirulina
reactor */
char    buffer[100];

/***********************************
```

```
      variables defined to get the history of
      the system for the input filters
***********************************/

extern double ALPHAFILTERcxa;
extern double ALPHAFILTERnitrate;
extern double ALPHAFILTEREb;
extern double ALPHAFILTERpH;

extern int SAMP_SPIRU;

int first_time_spiruline;
#endif
```

Help.h

```
#ifndef __help_def
#define __help_def
/***********************************************************************

NAME            HELP.H

AUTHOR          Pedro Pons

DESCRIPTION     Help for the keyboard commands

UPDATES
        20-09-95

***********************************************************************/

#include "results.h"

/*void help(ScreenViews);*/
void help(void);

#endif
```

Melfct.h

```
#ifndef __melfct_def
#define __melfct_def

/*******************************************************************
        NAME            MELFCT.H
        AUTHOR          BINOIS C
        DESCRIPTION
                general functions listing file (prototyping).
        UPDATES
                10-03-93
*******************************************************************/

#ifndef ADERSA
#include <graph.h>
#include <process.h>
#endif
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <math.h>

#include "melissa.h"
#include "userdef.h"

extern unsigned long lastsamp_sp; /* last sampling for spirulina
reactor */
extern unsigned tsamp_sp;     /* sampling interval for spirulina
reactor*/
extern unsigned long last_display_sp;

extern unsigned long lique_lastsamp; /* last sampling for reactor 2*/
extern unsigned lique_tsamp;     /* sampling interval for reactor 2 */
extern unsigned long lique_last_display;

extern unsigned long nitri_lastsamp; /* last sampling for reactor 3*/
extern unsigned nitri_tsamp;     /* sampling interval for reactor 3 */
extern unsigned long nitri_last_display;

extern unsigned long rhodo_lastsamp; /* last sampling for reactor 4*/
extern unsigned rhodo_tsamp;     /* sampling interval for reactor 4 */
extern unsigned long rhodo_last_display;

extern unsigned long main_lastsamp; /* last sampling for reactor 4*/
extern unsigned main_tsamp;     /* sampling interval for reactor 4 */
extern unsigned long main_last_display;

extern long IntervalSpiru;
extern long IntervalLique;
extern long IntervalRhodo;
extern long IntervalNitri;

#endif
```

Melissa.h

```
#ifndef __melissa_def
#define __melissa_def

/*********************************************************************

      NAME            MELISSA.H

      AUTHOR          BINOIS C   (modified by FULGET N. ADERSA)

      DESCRIPTION
            General Declarations

      UPDATES
            20-09-95

*********************************************************************/



/*--------------------------------------
      constants for VARS
--------------------------------------*/
#define TAG     128
#define CMD     255
#define UNDEF   0
#define NB_SAMP 0xFF          /* number of samples stored in val[ ] */


/*--------------------------------------
      structure for variables
--------------------------------------*/

typedef struct _vars {
      char    name[9];          /* tag name              */
      char    file[12];         /* file name             */
      int     type;             /* rd_gps/set_cmd return code
*/
      unsigned char tag_cmd;    /* TAG or CMD            */
      unsigned int dev_num;     /* controller number     */
      double  value;            /* current value         */
      int     i;                /* pointer on last value entered in
val[] */
      double  val[NB_SAMP+1]; /* previous values       */
      double  min;
      double  max;
      double  sp;               /* set point for LOOP       */
      double  out;              /* out value for LOOP       */
      char    unit[5];          /* unit for analog values   */
      char    update;           /* ON when structure updated*/
} VARS;

/*--------------------------------------
      structure for opened GPS files
--------------------------------------*/

typedef struct _gps_file{
      char    file[15];             /* file name                */
      int     handler;              /* handler of gps file      */
      int     rank;                 /* rank of the gps file     */
      struct  _gps_file   *next;   /* next opened gps file     */
} GPS_FILE;
```

```
/*------------------------------------
      structure for reactor state
------------------------------------*/

typedef struct  _react{
      double  Cxa;    /* in mg/l       */
      double  Cno3;   /* in mg/l       */
      double  temp;   /* in degre C    */
      double  press;  /* in            */
      double  Eb;     /* in W/m2       */
      double  Fr;     /* in W/m2       */
      double  rxa;    /* in mg/l/h     */
      double  rn;     /* in mg/l/h     */
      double  ro2;    /* in mg/l/h     */
} REACT;

/*------------------------------------
      general constants
------------------------------------*/

#define TSAMP_MAIN   60         /* sampling interval in secondes
*/
#define SYNCHRO  1
#define ERROR_SPEED 0.01        /* max error for the model      */
#define VOLUME_LIGHT 3.900      /* illuminated volume (in l)    */
#define VOLUME_TOTAL 7.000      /* total volume (in l)          */

/*------------------------------------
      Model control constants
------------------------------------*/

#define CAL_FR  10
#define CAL_ACT  20


/*------------------------------------
      Mathematical constants
------------------------------------*/

#define PI      3.14159265359


/*------------------------------------
      colours
------------------------------------*/

#define BLACK   0
#define BLUE    1
#define GREEN   2
#define CYAN    3
#define RED     4
#define MAGENTA 5
#define BROWN   6
#define WHITE   7

/*------------------------------------
      ADERSA constants
------------------------------------*/

#define DT      30      /* sampling period of PFC       */
#define NHC     5       /* coincidence point in DT      */
#define LAMBDA  0.75    /* reference trajectory dynamic */
```

```
#define DFR      10.     /* radiant flux increment W/m2   */
#define FR_MIN   10.     /* min constraint on Fr W/m2     */
#define FR_MAX   400.    /* max constraint on Fr W/m2     */
#define DQ       0.1     /* flow variation                */
#define CXA_MIN  250.    /* min constaint on cxa mg/l     */
#define CXA_MAX  1500.   /* max constaint on cxa mg/l     */



/* =====================================
      Definitions for displaying
      multiples reactors
===================================== */

typedef enum _reactors { principal,
                   spirulina,
                   gspirulina,
                   igspirulina,
                   nitrifying,
                   gnitrifying,
                   ignitrifying,
                   reactor_3,
                   reactor_4,
                   help_1,
                   status,
                   filemanagement,
                   simulation1,
                   simulation2,
                   simulation3,
                   simulationParams
                   } ScreenViews;

/*-------------------------------------
      ADERSA constants
-------------------------------------*/

#define DT      30             /* sampling period of PFC        */
#define DT_NITROGEN      30      /* sampling period of PFC        */
#define NHC      5       /* coincidence point in DT       */
#define LAMBDA   0.75    /* reference trajectory dynamic  */
#define DFR      10.     /* radiant flux increment W/m2   */
#define FR_MIN   10.     /* min constraint on Fr W/m2     */
/*#define FR_MAX  32000.    /* max constraint on Fr W/m2     */
#define DQ       0.1     /* flow variation                */
#define CXA_MIN  250.    /* min constaint on cxa mg/l     */
#define CXA_MAX  1500.   /* max constaint on cxa mg/l     */

extern int SAMP_GLOB;

/* =================================
    output file definitions.
   ================================= */

#define CONFIG_FILE "melissa.cfg"

/*#define __Monitoring    /* Just for read only */

#endif
```

Pargene.h

```
#ifndef _Pargene_H_
#define _Pargene_H_
/*
        pargene.h

   general parameters
   ---------------- */

# define nT 5000        /* number max of time steps (modifiable by the
user) */
# define nZ 1000        /* number max of integration steps along the
radius (modifiable by the user) */
# define nsig 9         /* number of compounds of the model (non
modifiable) */
# define ncoef 5  /* number of coefficients of the global formula (non
modifiable) */
# define ncomp 6  /* number of compounds of the biomass(non modifiable)
*/

/* parameters of the integration methods */
#define nstep      500.   /* number of integration steps along the
radius */
#define sim_dt         0.5    /* sampling period (h) */
#define tsim       500.  /* Hours of duration on the simulation */
#endif
```

Parmodel.h

```
#ifndef _parmodel_H_
#define _parmodel_H_
/*
      parmodel.h

   first principles model parameters of photoautotroph compartment
   ---------------------------------------------------------------- */

#define RT          .045    /* reactor radius (m) */
#define wiv         0.52      /* working illuminated volume
(dimensionless) */
#define Ea          872.    /* global absorption mass coefficient (m2/kg)
*/
#define Es          200.    /* global scattering mass coefficient (m2/kg)
*/
#define mupM        .45     /* max growth rate for biomass (1/h) */
#define mupMEPS     1.852   /* max growth rate for biomass phycocyanins
(1/h) */
#define Kj          20.     /* half satur. cste for energy to biomass
(W/m2) */
#define KjEPS       750.    /* half satur. cste for energy to EPS (W/m2)
*/
#define Fmin        1.      /* min incident radiant energy flux (W/m2) */

#define KN          5.3e-3 /* half satur. cste for nitrate conc. (kg/m3)
*/
#define KS          2.5e-4 /* half satur. cste for sulfate conc.(kg/m3)
*/
#define KPC         .06     /* half satur. cste for phycocyanin
conc.(kg/m3) */
#define zCH         .01     /* mass biotic fraction of CH (dimensionless)
*/
#define zPC         .162    /* mass biotic fraction of PC (dimensionless)
*/
#define zP          .684   `/* mass biotic fraction of P (dimensionless)
*/
#define YNXA        .516    /* mass conver. yield of NO3 in XA
(dimensionless) */
#define YSXA        .022    /* mass conver. yield of SO4 in XA
(dimensionless) */
#define YSEPS       .049    /* mass conver. yield of SO4 in EPS
(dimensionless) */
#define qq          .55     /* coefficient of proportionality
(dimensionless) */

#define Mab         23.096 /* molar mass of active biomass     (g/C_mole)
*/
#define Mgly        25.07  /* molar mass of glycogen           (g/C_mole)
*/
#define Meps        29.33  /* molar mass of exopolysaccharide (g/C_mole)
*/


#endif
```

Results.h

```
#ifndef __results_def
#define __results_def
/*********************************************************************

        NAME                RESULTS.H

        AUTHOR              Pedro Pons

        DESCRIPTION
                            Routines for displaying results.

        UPDATES
                    96.05.03

*********************************************************************/


#include <stdio.h>
#include <graph.h>
#include <string.h>
#include <time.h>
#include <ctype.h>
#include <errno.h>
#include <dos.h>

#include "ctrlspir.h"
#include "ctrlnitr.h"
#include "ctrlliqu.h"
#include "ctrlrhod.h"
#include "userdef.h"
#include "melfct.h"
#include "melissa.h"
#include "alarms.h"

int result(ScreenViews *display_reactor, int a);
result_spirulina();

int init_output_file(FILE*, char*);
int init_log_file(void);
void error_log_file (char*);
void log_file (char*);

extern int SAMP_SPIRU,
           SAMP_NITRI,
           SAMP_RHODO,
           SAMP_LIQUE;
extern int diskfull,
           diskwarning;


#endif
```

Screen.h

```
#ifndef _screen_h
#define _screen_h
/*********************************************************************

NAME            SCREEN.H

AUTHOR          Pedro L. Pons

DESCRIPTION
        screen and graphic declaration file

UPDATES
        960624

*********************************************************************/

#define NUMVARSTOPLOT 9


#define Simulation1FileName "/gpssim/compo.res"
#define Simulation2FileName "/gpssim/conc.res"
#define Simulation3FileName "/gpssim/glob.res"

typedef struct _datagraph
    {
    int activ[NUMVARSTOPLOT];   /* Is a whished variable for the
graphical representation ? */
    float v[NUMVARSTOPLOT],     /* inputs */
          y[NUMVARSTOPLOT],     /* y per input */
          max[NUMVARSTOPLOT],   /* Maximum possible */
          min[NUMVARSTOPLOT];   /* Minimum desired */
    int colors[NUMVARSTOPLOT];
    char name[NUMVARSTOPLOT][50];  /* Names for the variables */
    void (*initialisation) (void);
    char title[100];
    char filename[4];
    } datagraph;


#endif
```

Simfile.h

```
#ifndef _simFiles_h
#define _simFiles_h

/* Here are defined the names of the files that have the paràmeters of
the simulation.
*/


#define FCr0 "/gpssim/cr0.txt"
#define FFRH "/gpssim/frh.txt"
#define FdilH "/gpssim/dilh.txt"
#define FCiH "/gpssim/cih.txt"
#define FCrH "/gpssim/crh.txt"
#define FmasbioH "/gpssim/fmasbioh.txt"
#define FchonspH "/gpssim/chonspH.txt"


#endif
```

UserDef.h

```
#ifndef _USERDEF_H
#define _USERDEF_H
/**************************************************************************
********

NAME       USERDEF.H

AUTHORS       (C) TOPTOOLS 1988,1989,1990,1991,1992

DESCRIPTION

   INDUSTAR General Purpose Station - User Include File


**************************************************************************
*******/


/* -----------------------------------
   Miscellaneous constants definition
---------------------------------- */

#define   LGCT       9              /* key length
*/
#define ACTIV        1              /* command is active
*/
#define INACTIV      0              /* command is inactive
*/

#define ON           1              /* ON  value  for digital commands
*/
#define OFF          0              /* OFF value  "       "       "
*/


/* ---------------------------
   rd_gps() return codes
--------------------------- */

                                    /* Tag type
Structure    */
#define ISBIT        1              /* digital                  IDIG
*/
#define ISABUS       2              /* analog                   IBUSA
*/


/* ---------------------------
   set_cmd() return codes
--------------------------- */

                                    /* Cmd type
Structure    */
#define OSBIT       32              /* digital                  ODIG
*/
#define OSMILOOP    41              /* analog  (Micon loop)     OMLO
*/
#define OSREGUL     42              /* analog  (AB,TCS,PLS loop) OREGU
*/
#define OSREGIST    52              /* analog  (register)       OREGIS
*/
```

```
#define OSMIVD       61            /* digital  (Micon DV)       OMVD
*/
#define OSMILOC      71            /* analog   (Micon loc)      OMLOC
*/


/* ---------------------------
   gpserror values
--------------------------- */

#define ERDOS         1            /* DOS problem
*/
#define ENETW         2            /* network or mailbox
*/
#define INVALID_FGPS 3             /* invalid GPS file
*/
#define ETAGCMD       4            /* tag or command not found
*/
#define EVARCOD       5            /* invalid variable codification
*/
#define ECONV         6            /* floating point conversion
*/
#define ETAGTYP       7            /* unknown tag type
*/
#define EEQUIP        8            /* unknown PLC protocol
*/
#define ECMDTYP       9            /* unknown command type
*/
#define ENUMPLC      10            /* invalid device number
*/
#define ERPROTECT    11            /* protection key not found
*/
#define ENSECTOR     12            /* invalid record number
*/
#define ETYPVARCAL   13            /* not a computed variable
*/
#define EGDPREV      24            /* action on previous GD var not
over     */
#define EGDTYPVAR    25            /* not a GD variable
*/
#define EGDPOLLIS    26            /* POLLIS.TAB not found
*/
#define EGDCMDFAIL   27            /* failed command to GD
*/
#define ELOCAL       31            /* PLC in Local state (cmd
impossible)   */
#define ISMODAUTO    32            /* AUTO mode : change is impossible
*/
#define EPLSOVER     44            /* PLStar variable value out of
limits    */


/* ---------------------------
   Not selected command fields
--------------------------- */

#define FVALNOTUSED (float)0xF4240 /* 4bytes, not selected analog cmd
field */
#define IVALNOTUSED 64             /* 1byte,  not selected dig or loop
fld. */


/* -----------------------------------------
   Specific definitions for Micon equipments
```

```
----------------------------------------- */

#define MICLOCREM   151             /* local/remote
*/
#define MICCASCA    150             /* cascade
*/
#define MICAUTO     149             /* auto
*/
#define MICMANUAL   148             /* manual
*/
#define ISINCONFIG 155              /* Micon is starting configuration
*/


/* ---------------------------
   External declarations
---------------------------- */

extern int gpserror;                /* variable to house error codes
*/
extern int echonetw ;               /* network error code, when
applicable   */
extern char *gettags () ;           /* tag or command name (8-char
string)   */

unsigned int nbtags;        /* # tags in activated group
*/
unsigned int nbcommands;    /* # cmds in activated group
*/

/* -------------------------------------------------------------------
--------

                    STRUCTURES FOR READING TAGS

----------------------------------------------------------------------
----- */


/* -------------------------------------------
   Common header structure (tags and commands)
------------------------------------------------ */

typedef struct _iohead  {

   char      tag[LGCT];             /* tag or cmd
*/
   char      tag_name[31];          /* "      "  name
*/
   unsigned char tag_type;          /* tag or cmd type :
                                        = 1  digital input
                                        = 2  analog    "
                                        = 3  digital output
                                        = 4  analog       "       (loop)
                                        = 5   "           "
(register)
                                        = 6   "           "       Micon
VD
                                        = 7   "           "       Micon
LOC
...................................   */
```

```
   unsigned char  zone;             /* INDUSTAR C&C Station number
*/
   unsigned int   dev_num;          /* controller number
*/
   unsigned char  dev_type;         /* controller type :
                                       = 1  MICON
                                       = 2  JBUS-MODBUS
                                       = 3  STRUTHERS & DUNN
                                       = 4  ALLEN BRADLEY
                                       = 5  UNITELWAY
                                       = 8  TCS 6000
                                       = 9  TIWAY
                                       = 13 LAC
                                       = 14 PLStar TT
                                       = 15 Ghost Device TT

.................................    */

   unsigned char   log_can;         /*  logical channel number
*/

} IOHEAD;


/* ---------------------------
   Digital Input
--------------------------- */

typedef struct _idig {

   unsigned char val_state;         /* tag state  (0/1)
*/
   unsigned char act_state;         /* active state
*/
   char          alarm_tag[LGCT];   /* associated alarm tag
*/
   char          fault_tag[LGCT];   /* associated fault tag
*/
   char          progress_tag[LGCT]; /* associated in progress tag
*/

   char          var_nam[7] ;       /* TIWAY - process variable name
*/
   int           var_typ ;          /* TIWAY - process variable type
*/
   int           var_num ;          /* TIWAY - process variable #
*/

} IDIG;


/* ---------------------------
   Analog tag
--------------------------- */

typedef struct _ibusa {

   float         val;               /* value  (IEEE floating point)
*/
   float         scale;             /* scale
*/
   float         vl;                /* very low limit
*/
```

```
   float        l;                  /* low limit
*/
   float        h;                  /* high limit
*/
   float        vh;                 /* very high limit
*/
   char         unit[5];            /* unit
*/

   char         var_nam[7] ;        /* TIWAY - process variable name
*/
   int          var_typ ;           /* TIWAY - process variable type
*/
   int          var_num ;           /* TIWAY - process variable #
*/

} IBUSA;


/* ----------------------------------------------------
   Digital or Analog tag (without the header structure)
---------------------------------------------------- */

typedef union _u_inp {

   IDIG   d;                        /* when digital
*/
   IBUSA  bus;                      /* when analog
*/

} U_INP;


/* -------------------------------------
   Structure for the calls to rd_gps()
------------------------------------- */

typedef struct _igps {

   IOHEAD  h;                       /* common header structure
*/
   U_INP   i;                       /* according to tag type and PLC
*/

} IGPS;


/* ----------------------------------------------------------------------
--------

               STRUTURES FOR GETTING COMMAND INFORMATION

----------------------------------------------------------------------
----- */


/* ---------------------------
   Digital command
--------------------------- */

typedef struct _odig {

   char   cmd_tag[LGCT];            /* tag name associated to the
command    */
```

```
   char    st_cmd;                  /* tag  state (0/1)
*/
   char    as_cmd;                  /* active state
*/
   char    cmd_file;                /* file number for ALLEN BRADLEY
*/
   int     cmd_typ ;                /* TIWAY
*/

   char    plcloc_tag[LGCT];        /* local/remote tag
*/
   char    st_plcloc;               /* local(0)/remote(1) tag state
*/
   char    as_plcloc;               /* active state
*/

   char    alarm_tag[LGCT];         /* alarm tag associated to the
command    */
   char    st_alarm;                /* state alarm tag (0/1)
*/
   char    as_alarm;                /* active state
*/

   char    fault_tag[LGCT];         /* fault tag
*/
   char    st_fault;                /* state fault tag (0/1)
*/
   char    as_fault;                /* active state
*/

   char    instart_tag[LGCT];       /* in progress tag
*/
   char    st_instart;              /* state in progress tag (0/1)
*/
   char    as_instart;              /* active state
*/

   char    inhalt_tag[LGCT];        /* in halt tag
*/
   char    st_inhalt;               /* state in halt tag (0/1)
*/
   char    as_inhalt;               /* active state
*/

   char    localcmd_tag[LGCT];      /* local command tag (element)
*/
   char    st_localcmd;             /* state local command tag (0/1)
*/
   char    as_localcmd;             /* active state
*/

   char    cmdtype;                 /* command type = 0,1,2,3
*/
   char    mult_tab;                /* not used by the G.P.S.
*/

   unsigned devon_adr;              /* PLC bit state adresse ON
*/
   char     as_devon;               /* active state for ON
*/
   char     mask_on;                /* bit mask for ALLEN BRADLEY
*/
```

```
   unsigned devoff_adr;              /* PLC bit state addresse OFF
*/
   char     as_devoff;               /* active state for OFF
*/
   char     mask_off;                /* bit mask for ALLEN BRADLEY
*/

   char    ntfdc;                    /* table # for dig command
*/
   unsigned ad_w;                    /* word address
*/
   char    nton;                     /* table # for address on
*/
   unsigned adw_on;                  /* word address  on
*/
   unsigned adb_on;                  /* bit  address  on
*/


} ODIG;


/* ---------------------------
   Analog command (register)
--------------------------- */

typedef struct _oregis {

   char     cmd_tag[LGCT];           /* tag name associated to the
command     */
   float    val;                     /* tag measure value
*/
   int      cmd_typ ;                /* TIWAY
*/

   unsigned char cmd_mod;            /* mode variable PLStar
*/
   char     locrem_tag[LGCT];        /* tag for local/remote state
*/
   char     state;                   /* PLC state Local(0)/Remote(1)
*/
   char     as_locrem;               /* active state Local/Remote
*/

   char     reg1_tag[LGCT];          /* tag register 1
*/
   float    reg1_val;                /* value register 1
*/
   float    reg1_min;                /* value mini  reg. 1
*/
   float    reg1_max;                /* value maxi  reg. 1
*/
   char     reg1_unit[5];            /* unit register 1
*/
   unsigned reg1_adrhx;              /* PLC adress (hexa) of register 1
*/
   unsigned char reg1_mod;           /* mode variable PLStar
*/
   char     reg1_file;               /* # file AB
*/
   int      reg1_typ ;               /* TIWAY
*/
   int      reg1_num ;               /* TIWAY
*/
```

```
   char     reg2_tag[LGCT];          /* tag register 2
*/
   float    reg2_val;                /* value register 2
*/
   float    reg2_min;                /* value mini  reg. 2
*/
   float    reg2_max;                /* value maxi  reg. 2
*/
   char     reg2_unit[5];            /* unit reg. 2
*/
   unsigned reg2_adrhx;              /* PLC adress (hexa) of register 2
*/
   unsigned char reg2_mod;           /* mode variable PLStar
*/
   char     reg2_file;               /* # file AB
*/
   int      reg2_typ ;               /* TIWAY
*/
   int      reg2_num ;               /* TIWAY
*/

   char     reg3_tag[LGCT];          /* tag register 3
*/
   float    reg3_val;                /* value register 3
*/
   float    reg3_min;                /* valeur mini reg. 3
*/
   float    reg3_max;                /* valeur maxi reg. 3
*/
   char     reg3_unit[5];            /* unit register 3
*/
   unsigned reg3_adrhx;              /* PLC adress (hexa) of register 3
*/
   unsigned char reg3_mod;           /* mode variable PLStar
*/
   char     reg3_file;               /* # file AB
*/
   int      reg3_typ ;               /* TIWAY
*/
   int      reg3_num ;               /* TIWAY
*/

} OREGIS;


/* ----------------------------
   Analog command (loop)
---------------------------- */

typedef struct _oregu {

   char     cmd_tag[LGCT];           /* tag name associated to the
command     */
   float    val;                     /* tag measure value
*/
   int      cmd_typ ;                /* TIWAY
*/
   int      tiloopnb ;               /* TIWAY
*/
   char     cmd_mod;                 /* mode variable PLStar
*/
```

```
   char     spt_tag[LGCT];           /* tag for setpoint
*/
   float    spt_val;                 /* setpoint value
*/
   float    spt_min;                 /* value mini  spt
*/
   float    spt_max;                 /* value maxi  spt
*/
   char     spt_unit[5];             /* unit for setpoint
*/
   unsigned spt_adr;                 /* adresse ALLEN B
*/
   unsigned spt_file;                /*  file number of spt value (A-B)
*/
   char     spt_mod;                 /* mode variable PLStar (cf.
PLStar)     */
   int      spt_typ ;                /* TIWAY
*/
   int      spt_num ;                /* TIWAY
*/


   char     mo_tag[LGCT];            /* tag for Manual Output
*/
   float    mo_val;                  /* M.O. value
*/
   float    mo_min;                  /* value mini  M.O.
*/
   float    mo_max;                  /* value maxi  M.O.
*/
   char     mo_unit[5];              /* unit  M.O.
*/
   unsigned mo_adr;                  /* adresse ALLEN B
*/
   unsigned mo_file;                 /* file number of M.O.
*/
   char     mo_mod;                  /* mode variable PLStar (cf.PLStar)
*/
   int      mo_typ ;                 /* TIWAY
*/
   int      mo_num ;                 /* TIWAY
*/

   char     stalo_tag[LGCT];         /* tag for loop status
*/
   char     stalo_mod;               /* status loop value
*/
   char     stalo_unit[5];           /* unit for loop status
*/
   unsigned stalo_adr;               /* adresse ALLEN B
*/
   unsigned stalo_file;              /* file number of loop status
*/
   char     stat_mod;                /* mode variable PLStar (cf.
PLStar)     */
   int      stalo_typ ;              /* TIWAY
*/
   int      stalo_num ;              /* TIWAY
*/

} OREGU;

/* --------------------------
```

```
   Analog command (Micon loop)
---------------------------- */


typedef struct _omlo {

   char     regutag[LGCT];          /* tag associated to the command
*/
   float    val;                    /* current value read in the
mailbox      */
   int      nloop;                  /* loop number
*/
   float    sp;                     /* setpoint value
*/
   float    spmin;                  /*    "      minimum value
*/
   float    spmax;                  /*    "      maximum    "
*/
   char     spunit[5];              /*  unit for setpoint
*/
   float    out;                    /* outpoint value
*/
   float    outmin;                 /*    "      minimum value
*/
   float    outmax;                 /*    "      maximum    "
*/
   char     outunit[5];             /* unit for outpoint
*/
   float    ratio;                  /* ratio value
*/
   float    ratiomin;               /*    "      mini
*/
   float    ratiomax;               /*    "      maxi
*/
   char     ratiounit[5];           /*  unit for ratio
*/
   float    bias;                   /* bias value
*/
   float    biasmin;                /*    "      mini
*/
   float    biasmax;                /*    "      maxi
*/
   char     biasunit[5];            /*   unit for bias
*/
   unsigned char state;             /* loop state (auto/manual/cascade)
*/

} OMLO;


/* ----------------------------
   Analog command (Micon LOC)
---------------------------- */

typedef struct _omloc {

   char     loctag[LGCT];           /* tag name associated to the
command     */
   float    val;                    /* current LOC value
*/
   int      nloc;                   /* LOC number
*/
   float    locmin;                 /*  minimum LOC value
*/
```

```
   float    locmax;                 /*  maximum  "    "
*/
   char     locunit[5];             /*  unit for LOC
*/

} OMLOC;


/* ---------------------------
   Digital command (Micon DV)
--------------------------- */

typedef struct _omvd {

   char     vdtag[LGCT];            /* tag name associated to the
command     */
   int      val;                    /* current Discrete Virtual value
*/
   int      nvd;                    /* Discrete Virtual number
*/

} OMVD;


/* ----------------------------------------------
   Digital or Analog command (without the header)
---------------------------------------------- */

typedef union _u_outp {

   ODIG     d;                      /* digital (common for all PLCs)
*/
   OMLO     ml;                     /* Loop: MICON
*/
   OREGU    l;                      /* Loop: ALLEN-BRADLEY, PLStar,
TIWAY    */
   OREGIS   r;                      /* Register: MODBUS, JBUS, LAC,
PLStar   */
                                    /* ...ALLEN-B, UNITELWAY, TIWAY
*/
   OMVD     vd;                     /* MICON VD
*/
   OMLOC    loc;                    /* MICON LOC
*/

} U_OUTP;


/* ----------------------------------------------------------
   Command structure for the calls to set_cmd() and wr_gps()
---------------------------------------------------------- */

typedef struct _ogps {

   IOHEAD   h;                      /* common header structure
*/
   U_OUTP   o;                      /* according to the PLC
*/

} OGPS;


/* ---------------------------------------------------------------------
--------
```

```
                    STRUCTURES TO SEND COMMANDS

-----------------------------------------------------------------------
----- */


/* -------------------------------------------------
   Sending a command to JBUS, MODBUS, LAC equipments
------------------------------------------------- */

typedef struct _usjbus {

   char   status_bit;          /* cmd state switch ACTIV/INACTIV
(default)   */
   char   action_bit;          /* new bit value (ON / OFF)
*/

   char   status_reg1;         /* cmd reg.1 switch ACTIV/INACTIV
(default)   */
   float  newreg1;             /* new  register 1 value
*/

   char   status_reg2;         /* cmd reg. 2 switch ACTIV/INACTIV
*/
   float  newreg2;             /* new  register 2 value
*/

   char   status_reg3;         /* cmd reg. 3 switch ACTIV/INACTIV
*/
   float  newreg3;             /* new register 3 value
*/

} USJBUS;


/* -----------------------------------
   Send a command to a Micon equipment
----------------------------------- */

typedef struct _usmloop {

   char   status_sp;           /* cmd setpoint switch
ACTIV/INACTIV(default) */
   float  val_sp;              /* new setpoint value
*/

   char   status_out;          /* cmd outpoint switch ACTIV/INACTIV
*/
   float  val_out;             /* new outpoint value
*/

   char   status_ra;           /* cmd ratio switch ACTIV/INACTIV
*/
   float  val_ra;              /* new ratio value
*/

   char   status_bi;           /* cmd bias switch ACTIV/INACTIV
*/
   float  val_bi;              /* new bias value
*/

   char   status_sta;          /* cmd state switch ACTIV/INACTIV
*/
```

```
   unsigned char mode;          /* new channel state value - use above
MICON
                                   definitions (MICLOCREM etc.)
*/

   char    status_loc;          /* cmd LOC switch ACTIV/INACTIV
*/
   float   val_loc;             /* new LOC value
*/

   char    status_vd;           /* cmd DV switch ACTIV/INACTIV
*/
   char    val_vd;              /* new DV value  ON/OFF
*/

} USMLOOP;


/* ---------------------------------------------------------------------
   Send a command to :
   Allen-Bradley, PLStar, UnitelWay, Reliance, TCS or TIWAY equipment
------------------------------------------------------------------- */

typedef struct _usalb {

   char    status_sp1;       /* cmd setpoint or reg 1 switch
ACTIV/INAC(default) */
   float   val_sp1;          /* new setpoint or reg 1 value
*/

   char    status_out2;      /* cmd outpoint or reg 2 switch
ACTIV/INACTIV    */
   float   val_out2;         /* new outpoint or reg 2 value
*/

   char    status_sta3;      /* cmd loop state or reg 3 switch
ACTIV/INACTIV  */
   float   val_sta3;         /* new loop state value or reg 3
*/

   char    status_bit;       /* cmd state switch ACTIV/INACTIV
*/
   char    val_bit;          /* new bit value  ON/OFF
*/

} USALB;


/* ------------------------------
   Send a command to the Mailbox
------------------------------ */

typedef struct _uscalc {

   char    status_bit;       /* state switch (ACTIV/INACTIV) for new bit
value*/
   char    val_bit;          /* new bit value
*/

   char    status_num;       /* state switch (ACTIV/INACTIV) for new ana
value*/
   float   val_num;          /* new analog value
*/
```

```
   char   destable[32];     /* list of zones to send the message
*/

} USCALC;


/* -----------------------------------
   Structure for the calls to wr_gps()
------------------------------------ */

typedef union _s_user {

   USJBUS    jb;            /* MODBUS,JBUS,LAC
*/
   USMLOOP   mic;           /* MICON
*/
   USALB     ab;            /* ALLEN
BRADLEY,PLStar,UNITELWAY,TCS,RELIANCE,TIWAY*/
   USCALC    cal;           /* Ghost Device TT
*/

} S_USER;


/* --------------------------
   GPS functions declarations
-------------------------- */

extern  int  open_gr(char *name_gr);
extern  int  activ_gr(int h_gr);
extern  int  rd_gps(char *mtag,struct _igps *_igps);
extern  int  rdm_gps(char *mtag,struct _igps *_igps,float *usvalc,int
nb);
extern  int  set_cmd(char *nom,struct _ogps *ogps);
extern  int  wr_gps(struct _ogps *stout,union _s_user *stuser);
extern  int  wrm_gps(struct _ogps *stout,float *usvalc,int nb);
extern  int  close_gr(int h_gr);
extern  char *gettags(unsigned int ii);
extern  int  garde(unsigned int n,unsigned int nsecs);
extern  int  wrm_mbx(struct _igps *_igps,float *usvalc,int nb);
extern  int  send_mess(struct _igps *_igps,union _s_user *user);
extern  void ses(void);

#endif
```

Vars.h

```
#ifndef __vars_def
#define __vars_def

/*********************************************************************

        NAME            VARS.H

        AUTHOR          Pedro Pons

        DESCRIPTION
                access to data via gps functions
                and variables management. Prototypes and
                define declaration.

        UPDATES
                20-09-95

*********************************************************************/


#ifndef ADERSA
#include <process.h>
#endif
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>

#include "melissa.h"
#include "userdef.h"
#include "ctrlspir.h"

#define MAX_CHECK_NETWORK 100

/* ************************************
    Definicions fetes per provar el sistema
   ************************************ */
/* #define _debugging_vars */

GPS_FILE *activ_grp_gps();
int my_interrupt();


#endif
```

C-files.

Alarms.c

```
/*******************************************************************

      NAME            ALARMS.C

      AUTHOR          Pedro Luis Pons

      DESCRIPTION
            ALARM management listing file

      UPDATES
        25-05-96, 11-05-97

*******************************************************************/

#include "alarms.h"
#include <stdlib.h>

int activated_alarms=1;

/*-------------------------------------
      alarm programm
-------------------------------------*/

void alarm_spiru()
      {
      long alval;
      int al=0;
    char bf[100], buffer[100];

    if (!activated_alarms) return;  /* To give the chance to
disconnect
                                              all the alarms.          */
      acq_alarm();
    if (alarm_micon1.value != 0)
        {
        al++;
      alval = (long) alarm_micon1_value.value;
/*      sprintf (buffer, "LOC-0121 : %li",alval);
     display_status (buffer); */
      wait_time(3);
        if (alval & 0x01)
            {
                  /* Alarm on P100 1 */
                  display_error ("Alarm in Micon 1");
                        rst_alarm1.value = 1;
            }
        if (alval & 0x02)
            {
                /* Alarm on P100 2 */
                display_error ("Alarm in Micon 2");
                      rst_alarm2.value = 1;
            }
        if (alval & 0x04)
            {
                /* Alarm on P100 3 */
                display_error ("Alarm in Micon 3");
                      rst_alarm3.value = 1;
            }
        if (alval & 0x08)
            {
                /* Alarm on P100 4 */
                display_error ("Alarm in Micon 4");
                      rst_alarm4.value = 1;
            }
        }

        if (Fr.value < (Fr.sp * 0.9) || Fr.value > (Fr.sp * 1.1))
            {
                /* Light is not between limits */
                display_error ("Light is not between limits");
                al++;
            }

      if (al) send_alarm();
      }

/*---------------------------------------
      alarms initialisation
--------------------------------------*/

init_alarm()
      {
      display_status("Initialisation of ALARMS ...");

      sprintf (rst_alarm1.name, "VD--0141");
      sprintf (rst_alarm2.name, "VD--0145");
      sprintf (rst_alarm3.name, "VD--0149");
      sprintf (rst_alarm4.name, "VD--0153");
      sprintf (rst_main_alarm.name, "VD--0155");
      sprintf (rst_micon1_alarm.name, "VD--0156");
      sprintf (rst_pressure.name, "VD--0144");
      /*sprintf (active_lamp.name, "VD--0146");*/
      sprintf (alarm_micon1.name, "LOC-0122");
      sprintf (alarm_micon1_value.name, "LOC-0121");

      wait_time(2);
      display_status(" ");
      }

/*---------------------------------------
      alarms acquisition
--------------------------------------*/

acq_alarm()
      {
      display_status("Acquisition of ALARMS ...");

      read_var(&rst_alarm1);
      read_var(&rst_alarm2);
      read_var(&rst_alarm3);
      read_var(&rst_alarm4);
      read_var( &rst_main_alarm );
```

```
        read_var( &rst_miconl_alarm );
        read_var( &rst_pressure);
        read_var( &alarm_miconl );
        read_var( &alarm_miconl_value);


        wait_time(2);
        display_status(" ");
        }

/*-----------------------------------
        alarms updating
-----------------------------------*/

send_alarm()
        {
        display_status("Updating ALARMS ...");

        write_var(&rst_alarm1);
        write_var(&rst_alarm2);
        write_var(&rst_alarm3);
        write_var(&rst_alarm4);
        write_var( &rst_main_alarm );
        write_var( &rst_miconl_alarm );
        write_var( &rst_pressure);
        /*write_var( &active_lamp );*/


        wait_time(2);
        display_status(" ");
        }

void alarm_lique(void)
{
}

void alarm_nitri(void)
{
}

void alarm_rhodo(void)
{
}
```

CtrlSpir.c

```
/*****************************************************************

      NAME            CTRLSPIR (EX: CONTROL.C)

      AUTHOR          BINOIS C     (modified by FULGET N. ADERSA)

      DESCRIPTION
            CONTROL PROGRAM listing file

      UPDATES
            20-09-95

*****************************************************************/

#include "ctrlspir.h"
#include <math.h>

double ALPHAFILTERcxa = 0.3;
double ALPHAFILTERnitrate = 0.3;
double ALPHAFILTEREb = 0.3;
double ALPHAFILTERpH = 0.3;


/*-------------------------------------
      mathematical model
--------------------------------------*/
#ifndef ADERSA
model(REACT *react)
#else
model(react)
REACT *react;
#endif
   {
    double  zpc=.135;
      double  zp=.57;
      double  zch=0.0085;
      double  zg=0.;
      double  za;
      double  Ea=871.;
      double  Es=167.;
      double  alpha,delta;
      double  Fr;
      double  R,R1,R2;
      double  z;
      double  jstep=0.01;
      double  pij,pijz;
      double  Kj=20;
      double  KN=5.3;
      double  muM=0.54;
      double  yn=0.42;

      double  coeft,coefN,Rmean;
      double z0, kstep;
      R=0.048;
      R1=0.0302;
      R2=0.02585;
```

```
/*   general parameters ----------------------------------*/

      za=zpc+zch;
      alpha=sqrt(za*Ea/(za*Ea+(1+zg)*Es));
      delta=(za*Ea+(1+zg)*Es)*react->Cxa/1000.*alpha*R;

/*   determination of the mean growth rate ---------------*/

      pij=0;
      Fr=react->Fr;
      z0 = 1.e-6 / R;
      kstep = (1. - z0) * jstep;
      for(z=z0;z<1.;z+=kstep)
            {
            if((z<R2/R)||(z>R1/R))
                  {

      pijz=Fr/z*2*cosh(delta*z)/(cosh(delta)+alpha*sinh(delta));
                  if(pijz>=1.)
                        {
                        pij+=z*pijz/(Kj+pijz);
                        }
                  }
            }
      Rmean=2.*kstep*muM*pij*zpc*react->Cxa*VOLUME_LIGHT/VOLUME_TOTAL;

/* temperature and nitrates correction */
/*
      coeft=0.8*exp(-pow((react->temp-35)/10,2))+0.2;
      coefN=react->Cno3/(KN+react->Cno3);
*/


/******  nitrate saturation  ********/
coefN=1;
/************************************/
/******  no temp correction  ********/
coeft=1;
/************************************/
      react->rxa=Rmean*coefN*coeft;
    react->rn=yn*react->rxa;
}

/*------------------------------------------------------------
        light calibration from UAB data on 16th September 1997
------------------------------------------------------------*/

#ifndef ADERSA
lightcal(REACT *react, int mode)
#else
lightcal(react, mode)
REACT *react;
int mode;
#endif
   {
      double  a_lamp = 6.27;
      double  b_lamp = -33.03;

      double  Fr;

      switch(mode) {

            case CAL_FR:
            {
```

```
            /*  FR determination -------------------------*/
            Fr = exp(a_lamp * act_lamp.value) + b_lamp;
            Fr = max(0.,Fr);
            react->Fr = Fr;
            break;
            }

            case CAL_ACT:
            {
            /*  light controller action determination ----*/
            act_lamp.sp = (log(react->Fr - b_lamp)) / a_lamp;
            break;
            }

            default:
            {
            display_error("Error in routine lightcal ...\n");
            display_error("*** Program terminated ***\n");
            exit(0);
            }
      }

    }



/*---------------------------------------
      copy structure reacta to reactb
--------------------------------------*/

#ifndef ADERSA
copy_react(REACT *reacta,REACT *reactb)
#else
copy_react(reacta,reactb)
REACT *reacta;
REACT *reactb;
#endif

      {
      reactb->Cxa=reacta->Cxa;
      reactb->Cno3=reacta->Cno3;
      reactb->temp=reacta->temp;
      reactb->press=reacta->press;

      reactb->Eb=reacta->Eb;
      reactb->Fr=reacta->Fr;
      reactb->rxa=reacta->rxa;
      reactb->rn=reacta->rn;
      reactb->ro2=reacta->ro2;
      }


/*---------------------------------------
      variables initialisation
--------------------------------------*/
init_vars_spirulina()
        {
        REACT   init_react;
        double  delta;
        double  prod;
        int     jj;

        display_status("Initialisation of variables ...");
```

```
/*  TAG and COMMAND name initialisation     */

        sprintf(cxa.name,"LOOP0107");
        sprintf(nitrate.name,"LOOP0103");
        sprintf(cal_nitrate.name,"DI--0125");

        sprintf(Eb.name,"LOOP0105");
        sprintf(Fr.name,"LOC-0128");
        sprintf(temperature.name,"LOOP0106");
        sprintf(pH.name,"LOOP0104");
        sprintf(act_pompe.name,"LOC-0154");
        sprintf(cal_pump.name,"LOC-0137");
        sprintf(act_lamp.name,"LOC-0146");

        sprintf(cons_prod_nom.name,"LOC-0150");
        sprintf(cons_prod_real.name,"LOC-0152");
        sprintf(qe_nom.name,"LOC-0151");
        sprintf(qe_real.name,"LOC-0153");
        sprintf(production.name,"LOC-0155");
        sprintf(prod_mod.name,"LOC-0156");

/*  Variables initialisation   */

        acq_vars_spirulina();

        init_react.Cxa=cxa.value;
        init_react.Cno3=nitrate.value;
        init_react.temp=temperature.value;
        init_react.Eb=Eb.value;
        lightcal(&init_react,CAL_FR);
        Fr.sp=init_react.Fr;
        write_var(&Fr);
        fill_struct_var(&cxa);

        cons_prod_real.sp=cons_prod_nom.value;
        write_var(&cons_prod_real);
        qe_real.sp=qe_nom.value;
        write_var(&qe_real);

/* Initialisation of the supervisor (for removal of the bias)  V2.2*/
        prod = cxa.value * qe_nom.value;
        sm_sup = prod;
        cons_sup = prod;

/* initialisation timer PFC */
        next_pfc_sp=DT;

        wait_time(1);
        _clearscreen (_GWINDOW);
        display_status(" ");
        }



/*-------------------------------------
     variables acquisition
-------------------------------------*/

acq_vars_spirulina()
    {
     display_status("Acquisition of variables ...");
```

```
        read_var(&cxa);

 /*   nitrate analyser calibration   */

        read_var(&cal_nitrate);
        if(!cal_nitrate.value)
              {
              read_var(&nitrate);
              }

        read_var(&Eb);
        read_var(&Fr);
        read_var(&temperature);
        read_var(&pH);
        read_var(&act_pompe);
        read_var(&cal_pump);
        read_var(&act_lamp);

        read_var(&cons_prod_nom);
        read_var(&cons_prod_real);
        read_var(&qe_nom);
        read_var(&qe_real);
        read_var(&production);
        read_var(&prod_mod);

        display_status(" ");
      }


/*           OJO VERSIO VELLA, pero porta els filtres */
/* acq_vars_spirulina()
      {
      char bf[200];
      double realval;

    display_status("Adquisition of variables ...");

    read_var(&cxa);
    if (!first_time_spiruline)
          {
      filter(cxa.name);
          }

/*  nitrate analyser calibration   */
/*
      read_var(&cal_nitrate);
      if(!cal_nitrate.value)
            {
            read_var(&nitrate);
      if (!first_time_spiruline)
                  {
          filter(nitrate.name);
                  }
            }

      read_var(&Eb);
    if (!first_time_spiruline)
            {
      filter(Eb.name);
            }
      read_var(&Fr);
      read_var(&temperature);
    read_var(&pH);
    if (!first_time_spiruline)
```

```
            {
      filter(pH.name);
            }
      read_var(&act_pompe);
    read_var(&cal_pump);

      read_var(&cons_prod_nom);
    read_var(&cons_prod_real);
    read_var(&qe_nom);
    read_var(&qe_real);
    read_var(&production);
    read_var(&prod_mod);


    display_status(" ");

      if (first_time_spiruline) first_time_spiruline = 0;
      }
*/
/*--------------------------------------
      commands updating
--------------------------------------*/

send_vars_spirulina()
    {
#ifndef __Monitoring
    display_status("Updating variables ...");

      write_var(&Eb);
      write_var(&Fr);
      write_var(&act_pompe);
      write_var(&cons_prod_real);
      write_var(&qe_real);
      write_var(&production);
      write_var(&prod_mod);
      write_var(&act_lamp);
#endif
    display_status(" ");
    }

/*--------------------------------------
      calculate the delta count during time t in minutes
--------------------------------------*/

#ifndef ADERSA
double diff_cpt(VARS *diff_var, int diff_time)
#else
double diff_cpt(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
        {
        int j;
        int i_samp,i_prev,nb_samp;
        double total_count;

        total_count=0;
        nb_samp=ceil(diff_time*60/(TSAMP_MAIN*SAMP_SPIRU)); /* Abans
TSAMP_SPIRULINA */
        for(j=0;j<nb_samp;j++)
                {
                i_samp=(diff_var->i-j)&NB_SAMP;
                i_prev=(i_samp-1)&NB_SAMP;
                total_count+= ( diff_var->val[i_samp]>=diff_var-
>val[i_prev]) ?
                diff_var->val[i_samp]-diff_var->val[i_prev] :
diff_var->val[i_samp];
                }
        return(total_count);
        }

/*---------------------------------------
      calculate the variable variation during time t in minutes
---------------------------------------*/



#ifndef ADERSA
double diff_var(VARS *diff_var, int diff_time)
#else
double diff_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
        {
        double dvar_dt;
        int nb_samp;

        nb_samp=ceil(diff_time*60/(TSAMP_MAIN * SAMP_SPIRU));
        dvar_dt=diff_var->val[diff_var->i]-diff_var->val[(diff_var->i
        -nb_samp)&NB_SAMP];
        return(dvar_dt);
        }


/*---------------------------------------
    calculate the average during time t in minutes
---------------------------------------*/

double average_var(VARS *diff_var, int diff_time)
    {
    int j;
    int i_samp,nb_samp;
    double average, aux1, aux2;
/*    char buffer[100];*/
    aux1 = diff_time * (double) 60;
    aux2 = (double) TSAMP_MAIN * (double) SAMP_SPIRU;
    average=0;
    nb_samp=ceil(aux1/aux2);
    for(j=0;j<nb_samp;j++)
        {
        i_samp=(diff_var->i-j)&NB_SAMP;
        average+=diff_var->val[i_samp];
        }
    average/= (double) nb_samp;
    return(average);
    }


/*---------------------------------------
        calculate the average^2 during time t in minutes
---------------------------------------*/


#ifndef ADERSA
double average2_var(VARS *diff_var, int diff_time)
#else
```

```
double average2_var(diff_var, diff_time)
VARS *diff_var;
int diff_time;
#endif
      {
      int j;
      int i_samp,nb_samp;
      double average;

      average=0;
    nb_samp=ceil((diff_time*60)/(TSAMP_MAIN * SAMP_SPIRU));
      for(j=0;j<nb_samp;j++)
            {
            i_samp=(diff_var->i-j)&NB_SAMP;
            average+=pow(diff_var->val[i_samp],2);
            }
    average/= (double) nb_samp;
    return(average);
    }

/*-------------------------------------
      fill val[i] with the current value
-------------------------------------*/

#ifndef ADERSA
fill_struct_var(VARS *fill_struct)
#else
fill_struct_var(fill_struct)
VARS *fill_struct;
#endif

      {
      int jj;

      for(jj=0;jj<=NB_SAMP;jj++)
            {
            fill_struct->val[jj]=fill_struct->value;
            }
      }


/*-------------------------------------
      fill val[i] with the current value and delta between each value
-------------------------------------*/

#ifndef ADERSA
fill_struct_cpt(VARS *fill_struct,double _delta)
#else
fill_struct_cpt(fill_struct,_delta)
VARS *fill_struct;
double _delta;
#endif

      {
      int jj,kk,ll;

      for(jj=0;jj<NB_SAMP;jj++)
            {
            kk=(fill_struct->i-jj)&NB_SAMP;
            ll=(kk-1)&NB_SAMP;
            fill_struct->val[ll]=fill_struct->val[kk]+_delta;
            }
      }
```

```
/*--------------------------------------
      calculate the slope of variable by the least mean square method
--------------------------------------*/

#ifndef ADERSA
double slope_var(VARS *slope_var,int diff_time)
#else
double slope_var(slope_var,diff_time)
VARS *slope_var;
int diff_time;
#endif
      {
      int ii,jj,kk;
      int nb_samp;
      double slope;
      double sumxi, sumyi, sumxiyi, sumxi2;

      sumxi=0;
      sumyi=0;
      sumxiyi=0;
      sumxi2=0;

    nb_samp=ceil(diff_time*60/(TSAMP_MAIN * SAMP_SPIRU));
      for(ii=0;ii<nb_samp;ii++)
            {
            jj=slope_var->i-ii;
            kk=(slope_var->i-ii)&NB_SAMP;
            sumxi+=jj;
            sumyi+=slope_var->val[kk];
            sumxiyi+=jj*slope_var->val[kk];
            sumxi2+=pow((double)jj,2);
            }
      slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-
sumxi*sumxi);
      return(slope);
      }


/*--------------------------------------
      calculate the slope of counter by the least mean square method
--------------------------------------*/

#ifndef ADERSA
double slope_cpt(VARS *slope_cpt,int diff_time)
#else
double slope_cpt(slope_cpt,diff_time)
VARS *slope_cpt;
int diff_time;
#endif
      {
      int ii,jj,kk,ll;
      int nb_samp;
      double slope;
      double sumxi, sumyi, sumxiyi, sumxi2;
      double raz_cpt;

      raz_cpt=0;
      sumxi=0;
      sumyi=0;
      sumxiyi=0;
      sumxi2=0;

    nb_samp=ceil(diff_time*60/(TSAMP_MAIN * SAMP_SPIRU));
```

```
        for(ii=0;ii<nb_samp;ii++)
                {
                jj=slope_cpt->i-ii;
                kk=(slope_cpt->i-ii)&NB_SAMP;
                ll=(kk-1)&NB_SAMP;
                sumxi+=jj;
                sumyi+=(slope_cpt->val[kk]-raz_cpt);
                sumxiyi+=jj*(slope_cpt->val[kk]-raz_cpt);
                sumxi2+=pow((double)jj,2);
                if(slope_cpt->val[ll]>slope_cpt->val[kk])
                        {
                        raz_cpt=slope_cpt->val[ll];
                        }
                }
        slope=nb_samp*(nb_samp*sumxiyi-sumxi*sumyi)/(nb_samp*sumxi2-
sumxi*sumxi);
        return(slope);
        }


        /**************************************************
        Sign
        **************************************************/
#ifndef ADERSA
 signe(double x)
#else
 signe(x)
double x;
#endif
        {
        x=(x<0) ? -1 : 1;
        return(x);
        }



/*----------------------------------------
        mathematical model for ADERSA
----------------------------------------*/

#ifndef ADERSA
double predimod(REACT react, double dil, int horiz)
#else
double predimod(react, dil, horiz)
REACT react;
double dil;
int horiz;
#endif
        {
/* react: current reactor model state     (Cxa,Fr,Cno3,temp)       */

/* dil      : dilution rate          (in h-1)          */
/* horiz: prediction horizon         (in number of DT) */
/* prod     : predicted production       (in mg/h)          */

        double v, prod;
        int k;

        v  = react.Cxa;       /* current biomass concentration */

          /* model integration (sampling period 1mn) */
        for(k=1;k<=horiz*DT;k++)
```

```
            {
                double Delta;

            react.Cxa=v;
            model(&react);
            Delta=1/60.0*(react.rxa-dil*v);
            v+=Delta;

            }

            prod=v*dil*VOLUME_TOTAL;
            return(prod);
}




/*---------------------------------------
      control programm
---------------------------------------*/

int control_spiru(ScreenViews active_reactor)
      {
      FILE *fp;
      int retval = 0;
      char buffer[300], buffer2[300];
      double  Fr1, Fr2, delfr;                              /*in
W/m2 */
      double  prod_ref,prod1,prod2,prod_max,prod_min;       /*in
mg/h */
      double  qe_max, qe_min;                               /*in 1/h
*/
      double  dil;                                          /*in h-1
*/
      double  cxa_moy , nit_moy;                            /*in
mg/l */
      REACT   react;
    double   cons_prod0 , cons_prod0_min , cons_prod0_max;
    double   dprod_min , dprod_max;

    sprintf (buffer, "Accessing spiruline data ...");
    _outtext (buffer);

    acq_vars_spirulina();

    #ifndef __Monitoring
    display_status("Control running ...");
    #endif

      if(!(next_pfc_sp--))
                {
                /* control PFC algorithm */

                    /* biomass concentration */
                cxa_moy=average_var(&cxa,10);
                nit_moy=average_var(&nitrate,10);

                    /* production calculation  */
                production.sp=cxa_moy*qe_real.value;

                    /* reactor state */
                    react.Cno3=nit_moy;
                    react.temp=temperature.value;
                    react.Cxa=cxa_moy;

                    /* flow and production constraints*/
                    qe_max=qe_nom.value*(1+DQ);
                    qe_min=qe_nom.value*(1-DQ);
                    prod_max=qe_max*CXA_MAX;
                    prod_min=qe_min*CXA_MIN;

                         /* feasible production setpoint calculation*/

cons_prod_real.sp=max(prod_min,min(prod_max,cons_prod_nom.value));

                         /* real flow setpoint  and corresponding
dilution rate*/
                    qe_real.sp=qe_nom.value;
                    if(cons_prod_real.sp/CXA_MAX>qe_nom.value)

{qe_real.sp=min(qe_max,cons_prod_nom.value/CXA_MAX);}
                    if(cons_prod_real.sp/CXA_MIN<qe_nom.value)

{qe_real.sp=max(qe_min,cons_prod_nom.value/CXA_MIN);}
                    dil=qe_real.sp/VOLUME_TOTAL;

                    /* supervisor */
                    sm_sup = LAMBDA * sm_sup + (1. - LAMBDA) * cons_sup;
                    /* 1_ output of the supervisor */
                    cons_prod0 = cons_prod_real.sp - production.sp +
sm_sup;
                    /* 2_ carrying forward the absolute constraints FR_MIN
FR_MAX */
                    react.Fr = FR_MAX;
                    dprod_max = predimod(react,dil,NHC) - production.sp;
                    cons_prod0_max = production.sp + dprod_max/(1.-
pow(LAMBDA,NHC));
                    react.Fr = FR_MIN;
                    dprod_min = predimod(react,dil,NHC) - production.sp;
                    cons_prod0_min = production.sp + dprod_min/(1.-
pow(LAMBDA,NHC));
                    cons_prod0 =
max(cons_prod0_min,min(cons_prod0_max,cons_prod0));
                    cons_sup = cons_prod0;          /* OJO afegida despres
del error */

                    /* reference trajectory */
                    prod_ref=cons_prod0-pow(LAMBDA,NHC)*(cons_prod0-
production.sp);

                    /*first scenario */
                    Fr1=Fr.value;
                    react.Fr = Fr1;
                    prod1=predimod(react,dil,NHC);

                    /* second scenario */
                    delfr=DFR*signe(cons_prod0-production.sp);
                    Fr2=Fr1+delfr;
                    react.Fr = Fr2;
                    prod2=predimod(react,dil,NHC);

                    /* Fr calculation */
                    Fr.sp=Fr.value+(prod_ref-prod1)/(prod2-prod1)*delfr;

                    /* constraints on Fr */
```

```
            Fr.sp=max(FR_MIN,min(FR_MAX,Fr.sp));

            /* light action sended to output of P100 controller */
            react.Fr=Fr.sp;
            lightcal(&react,CAL_ACT);

            /* pump setpoint sended to P100 controller */
            act_pompe.sp=qe_real.sp/cal_pump.value;

            /* model output calculation */
            prod_mod.sp = predimod(react,dil,1);

            next_pfc_sp=DT;
            retval = 1;
            }
  send_vars_spirulina();

sprintf (buffer, " done\n");
_outtext (buffer);

  return retval;
  }
```

CtrlNitr.c

```
/*********************************************************************

      NAME            CTRLNITR.C

      AUTHOR          Pedro L. Pons

      DESCRIPTION     Nitrogen reactor control program


      UPDATES
            25-05-96

*********************************************************************/

#include "ctrlnitr.h"

void acq_vars_nitrogen(void);

/*---------------------------------------
      variables initialisation
---------------------------------------*/


init_vars_nitrogen()
      {
      REACT   init_react;
      double  delta;
      int     jj;

      first_time_nitrogen = 1;          /* Set for the first filter
entry */
      display_status("Initialisation of variables ...");

      /* Initialisation of filter parameters */


/*   TAG and COMMAND name initialisation     */

    sprintf(NitrDo.name,"LOOP0303");
    sprintf(NitrpH.name,"LOOP0304");
    sprintf(NitrTemp.name,"LOOP0307");
    sprintf(NitrPressure.name,"LOOP0307"); /* Should be 0308 */
    sprintf(NitrAmCon.name,"LOOP0404");
    sprintf(NitrNitrCon.name,"LOOP0407");

/*   Variables initialisation   */

    acq_vars_nitrogen();

    next_pfc_nt=DT_NITROGEN;

      wait_time(1);

      display_status(" ");
    }


/*---------------------------------------
      variables acquisition
---------------------------------------*/
```

```
void acq_vars_nitrogen(void)
      {
      char bf[200];

      display_status("Acquisition of variables ...");


/*    read_var(&NitrDo);
    read_var(&NitrpH);
    read_var(&NitrTemp);
    read_var(&NitrPressure);
    read_var(&NitrAmCon);
    read_var(&NitrNitrCon);         OJO Prova */

    display_status(" ");
    }


/*--------------------------------------
      commands updating
--------------------------------------*/

send_vars_nitri()
      {
      }

/*--------------------------------------
      control programm
--------------------------------------*/

int control_nitrogen(ScreenViews active_reactor)
      {
      int retval=0;

    sprintf (buffer, "Accessing nitrogen data ...");
    _outtext (buffer);

      acq_vars_nitrogen();

      if (next_pfc_nt-- == 0)
            {
            display_status("Apling model for nitrogen compartment.");
        wait_time(2);
        display_status(" ");
            next_pfc_nt = DT_NITROGEN;
        retval = 1;
            }
    sprintf (buffer, " done\n");
    _outtext (buffer);

      return retval;
}
```

CtrlLiqu.c

```
/*********************************************************************

      NAME           CTRLLIQU.C

      AUTHOR         Pedro L. Pons

      DESCRIPTION    Lique reactor control program


      UPDATES
            25-05-96

*********************************************************************/

#include "ctrlliqu.h"

int next_pfc_lq = 0;


/*--------------------------------------
      variables initialisation
--------------------------------------*/

/*--------------------------------------
      variables acquisition
--------------------------------------*/

acq_vars_lique()
      {
      }

/*--------------------------------------
      commands updating
--------------------------------------*/

send_vars_lique()
      {
      }

/*--------------------------------------
      control programm
--------------------------------------*/

int control_lique(ScreenViews active_reactor)
      {
      int retval=0;

    sprintf (buffer, "Accessing lique data ...");
      _outtext (buffer);

    wait_time(1);

    sprintf (buffer, " done.\n");
    _outtext (buffer);

      return retval;
      }
```

CtrlRhod.c

```
/**********************************************************************

      NAME              CTRLRHOD.C

      AUTHOR            Pedro L. Pons

      DESCRIPTION       Rhodo reactor control program


      UPDATES
            25-05-96

**********************************************************************/

#include "ctrlrhod.h"

int next_pfc_rh = 0;

/*-------------------------------------
      variables initialisation
-------------------------------------*/

/*-------------------------------------
      variables acquisition
-------------------------------------*/

acq_vars_rhodo()
      {
      }

/*-------------------------------------
      commands updating
-------------------------------------*/

send_vars_rhodo()
      {
      }

/*-------------------------------------
      control programm
-------------------------------------*/

int control_rhodo(ScreenViews active_reactor)
      {
      int retval=0;

    sprintf (buffer, "Accessing rhodobacter data ....");
    _outtext(buffer);
    wait_time(1);
    sprintf (buffer, " done\n");
    _outtext(buffer);
    wait_time(1);

      return retval;
      }
```

GpsFile.c

```
/*******************************************************************
      NAME            GPSFILE.C
      AUTHOR          BINOIS C
      DESCRIPTION
            management of gps files
      UPDATES
            25-03-93
*******************************************************************/

#ifndef ADERSA
#include <process.h>
#endif
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "melissa.h"
#include "userdef.h"

static GPS_FILE *gps;

/*-------------------------------------
      open all gps files declared in GPS.FIL
-------------------------------------*/

GPS_FILE *open_file_gps()
      {
      FILE    *fp;
      int     hl,i,file_rank;
      char    buffer[80],str[80];
      GPS_FILE    *gp,*gp_save;
      int     file_opened;

      file_opened=OFF;
      file_rank=1;
      display_status("opening GPS files ...");
      fp=fopen("GPS.FIL","r");
      if(fp==NULL)
            {
            display_error("Could not open file GPS.FIL ... *** program
stopped ***");
            exit(0);
            }
      while(fscanf(fp,"%s",buffer)!=EOF)
            {
            if(sscanf(buffer,"BASE:%s",str)!=1)
                  {
                  display_error("Syntax error in file GPS.FIL ... ***
program stopped ***");
                  fclose(fp);
                  exit(0);
                  }
            for(i=1;;i++)
                  {
                  if (strlen (str)>6) {
                              sprintf(buffer,"%s.GPS",str);
                              if (i>1) break;
                              }
                        else
                        sprintf(buffer,"%s%02d.GPS",str,i);
```

```
                hl=open_gr(buffer);
                if(hl==-1)
                        {
                        break;
                        }
                gp=(GPS_FILE *)malloc(sizeof(GPS_FILE));
                if(gp==NULL)
                        {
                        sprintf(buffer,"Can't allocate memory for
%s%02d.GPS *** program stopped ***",str,i);
                        display_error(buffer);
                        exit(0);
                        }
                if(!file_opened)
                        {
                        gp->next=gp;
                        gp_save=gp;
                        }
                sprintf(gp->file,"%s",buffer);
                gp->handler=hl;
                gp->rank=file_rank;
                gp->next=gp_save->next;
                gp_save->next=gp;
                gp_save=gp;
                file_opened=ON;
                file_rank++;
                sprintf(buffer,"file %s opened ...\n",gp->file);
                _outtext(buffer);
                wait_time(1);
                }
        }
        if(!file_opened)
                {
                display_error("No GPS file found ... *** program terminated
***");
                exit(0);
                }
        fclose(fp);
        gps=gp;
        return(gp);
        }

/*-------------------------------------
        active one group using GPS_FILE struct
-------------------------------------*/

GPS_FILE *activ_grp_gps()

        {
        GPS_FILE        *gp;
        char    buffer[80];
        int     ret_activ_gr;

        gp=gps->next;
        ret_activ_gr=activ_gr(gp->handler);
        if(ret_activ_gr==-1)
                {
                sprintf(buffer,"Can't activate file %s ... *** program
stopped ***",gps->file);
                display_error(buffer);
                exit(0);
                }
        gps=gp;
        display_activ_group(gps);
```

```
      return(gp);
      }


/*--------------------------------------
      close all groups using GPS_FILE struct
--------------------------------------*/

void close_grp_gps()

      {
      GPS_FILE    *gp;
      char    buffer[80];
      int     file_rank, ret_close, all_closed;

      file_rank=gps->rank;
      all_closed=OFF;
      while(!all_closed)
            {
            ret_close=close_gr(gps->handler);
            if(ret_close==-1)
                  {
                  sprintf(buffer,"\nCan't close file %s ...\n",gps-
>file);
                  display_error(buffer);
                  error_gps();
                  }
            else
                  {
                  sprintf(buffer,"\nFile %s closed",gps->file);
                  _outtext(buffer);
                  wait_time(1);
                  }
            gps=gps->next;
            if(gps->rank==file_rank)
                  {
                  all_closed=ON;
                  }
            }
      display_no_activ_group();
      }
```

Graph.c

```
#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <dos.h>
#include <bios.h>
#include <time.h>
#include <graph.h>

static double x=40;
static int Xo=60, Yo=245;
static int Heigth = 200;     /* Heigth of the graphic */
static int Length = 350;     /* Length of the graphic */
#define NUMVARSTOPLOT 9


typedef struct _datagraph
    {
    int activ[NUMVARSTOPLOT];    /* Is a whished variable for the
graphical representation ? */
    float v[NUMVARSTOPLOT],      /* inputs */
          y[NUMVARSTOPLOT],      /* y per input */
          max[NUMVARSTOPLOT],    /* Maximum possible */
          min[NUMVARSTOPLOT];    /* Minimum desired */
    int colors[NUMVARSTOPLOT];
    char name[NUMVARSTOPLOT][30];  /* Names for the variables */
    } datagraph;


static datagraph data;


void InitializeSpPlot(void)
{
FILE *fp;
int i;
int v;

fp = fopen ("graph.cfg","rt");
if (fp==NULL)
    {
    data.activ[0] = 0;
    data.activ[1] = 0;
    data.activ[2] = 0;
    data.activ[3] = 0;
    data.activ[4] = 0;
    data.activ[5] = 0;
    data.activ[6] = 0;
    data.activ[7] = 0;
    data.activ[8] = 0;
    return;
    }
for (i=0; i < NUMVARSTOPLOT && ! feof (fp); i++)
    {
    v=fscanf (fp, "%i %i %i %30s", &data.activ[i], &data.colors[i],
&data.min[i], data.name[i]);
    if (v != 4)
        {
        data.activ[0] = 0;
        data.activ[1] = 0;
        data.activ[2] = 0;
        data.activ[3] = 0;
        data.activ[4] = 0;
        data.activ[5] = 0;
        data.activ[6] = 0;
        data.activ[7] = 0;
        data.activ[8] = 0;
        i=NUMVARSTOPLOT;
        }
    }
fclose(fp);
}


void DrawLegend (void)
{
int used=0,
    base=7, i;

for (i=0; i < NUMVARSTOPLOT; i++)
    {
    if (data.activ[i])
        {
        _settextposition (base+used, 60);
        _settextcolor(data.colors[i]);
        _outtext(data.name[i]);
        _settextcolor(15);
        used++;
        }
    }


}


void DrawScale(int i)
{
char buffer[100];

_settextcolor(data.colors[i]);
_settextposition(3,1);
_outtext("        ");
_settextposition(3,1);
sprintf (buffer, "%4.2f", data.y[i]);
_outtext(buffer);
_settextcolor(15);

_setcolor(3);
_moveto(Xo-4,Yo-Heigth);
_lineto(Xo+4,Yo-Heigth);
_moveto(Xo-4,Yo-Heigth-1);
_lineto(Xo+4,Yo-Heigth-1);

}


void DrawAxesSP (int np, long beg, long end)
{
char cad[100];

_setcolor (3);
_moveto (Xo, Yo);
_lineto (Xo+(Length*1.1), Yo);
_moveto (Xo, Yo);
```

```
_lineto (Xo, Yo - (Heigth *1.1));

_settextposition (1, 30);
_outtext ("Spirulina compartment");
/*_settextposition ((Yo+9)/8, (Xo + Length/2)/8);
_outtext ("Time (days) ...");
_settextposition(((double) Xo - (double) 24 )/(double) 8, (Yo(double)
+(double) 18)/(double)8);
sprintf (cad,"%li", beg);
_outtext (cad);
_settextposition(((double)Yo+(double)18)/(double)8, (Xo (double)-
24(double))/8(double) );
sprintf (cad,"%li", end);
_outtext (cad);*/
}


int ReadDrawSP (char *name, int np)
{
FILE *fp;
char c[2], input[300];
char hora[6], cr;
int line=0;
int first=6;     /* N§ of variables to graph */
double delta;     /* pixel per point */
int i;

double loc;

delta = Length / (double) np;

c[1]='\0';
strcpy (input, "");

fp = fopen (name, "rt");
if (fp==NULL) return -1;
    else {
            read (&cr,1,1,fp);
            while (cr != '\n') fread (&cr, 1,1,fp);
            while (!feof(fp))
              {
              fread (&c[0], 1, 1, fp);
              if (ferror (fp)) return -1;
              if (c[0] == '\n')   /* let's read all a line ... */
                  {
                  line++;
                  sscanf (input, "%5c %f %f %f %f %f %f %f %f %f",hora,
&data.v[0], &data.v[1], &data.v[2], &data.v[3], &data.v[4],
&data.v[5], &data.v[6], &data.v[7], &data.v[8]);
                  for (i=0; i < 9; i++)
                    {
                      if (data.activ[i] ) {
                      if (!first) _moveto (x,data.y[i]);
                          else
                              {
                              _moveto(x, Yo - (((data.v[i]-
data.min[i])/data.max[i]) * Heigth) );
                              first--;
                              }
                      _setcolor (data.colors[i]);
                      data.y[i] = data.v[i];
                      data.y[i] = ((data.y[i]-data.min[i]) /
data.max[i]) * Heigth;
                      data.y[i] = Yo - data.y[i];
```

```
                            _lineto ((int)x+delta,data.y[i]);
                            }
                          }
                        x += delta;
                        strcpy (input, "");      /* Reinitialize input string
*/
                        }
                        else
                        {
                        strcat (input, c);
                        }

                    }

                fclose(fp);
                return line-2;     /* Return the n§ of points of the file
(last, first) */
                }
}


int DrawSP (int np, long beg, long end)
{
long k;
int r;
char cad[20];

x = Xo;

/*_moveto (Xo, Yo-Heigth);
_lineto (Length, Yo-Heigth);*/


for (k=beg; k <= end; k++)
    {
    sprintf (cad, "sp%li.out", k);
                  /* Test if file exist */
    r = ReadDrawSP(cad,np);
    if (r==-1)
            {
            return -1;
            }
    }
return 0;

}


int testfileexist (char *name)
{
FILE *fp;
char c[2],
     hora[6],
     input[300];

int i;
int line=0;

c[1] = '\0';

strcpy (input, "");
fp = fopen (name, "rt");
if (fp==NULL) return -1;
    else {
          while (!feof(fp))
            {
            fread (&c[0], 1, 1, fp);
            if (c[0] == '\n')
                {
                line++;
                if (line >1)
                    {
                    /* Check for maximum */
                    sscanf (input, "%5c %f %f %f %f %f %f %f %f
%f",hora,
                        &data.v[0], &data.v[1], &data.v[2],
&data.v[3],
                        &data.v[4], &data.v[5], &data.v[6],
&data.v[7],
                        &data.v[8]);
                    for (i=0; i < NUMVARSTOPLOT; i++)
                        {
                        if (data.v[i] > data.max[i]) data.max[i] =
data.v[i];
                        }
                    strcpy (input,"");
                    }
                }
                else {
                strcat(input, c);
                }
            }

          fclose(fp);
          return line-2;     /* Return the n§ of points of the file
(last, first) */
        }
}


int calcule_between (long beg, long end)
{
long k;
long lines=0, lf;
char cad[100], cad2[100];

if (abs (end - beg) > 15) return -1;     /* Only graphics for 15 days
*/

_outtext ("\n\n");

for (k=beg; k <= end; k++)
    {
    sprintf (cad, "sp%li.out", k);
                /* Test if file exist */
    lf = (long) testfileexist(cad);
    if (lf==-1) {
                return -1;
                }
        else
        lines = lines + lf;
    }

return lines;
}


int drawgraph_spiruline (void)
```

```
{
char cad[100];
long begin, end, nlines;
time_t ltime;
struct  tm  *dt;

_settextposition(1,10);
_outtext ("Initial date : (yymmdd) ");
scanf ("%6s", cad);

begin = atol(cad);

time(&ltime);
dt=localtime(&ltime);
dt->tm_mon++;
sprintf(cad,"%02d%02d%02d",dt->tm_year,dt->tm_mon,dt->tm_mday);
end = atol(cad);

InitializeSpPlot();
DrawLegend();

nlines = calcule_between (begin, end);  /* Get the number of points to
draw */
if ( nlines > 0)
        {
        DrawAxesSP (nlines, begin, end);
        if (DrawSP (nlines, begin, end)==-1)
            {
            _outtext ("Error during drawing process!!");
            }
        }
        else
        {
        _outtext ("Unable to draw graph.");
        }


}


main()
{

_setvideomode(_VRES16COLOR);

_clearscreen (_GWINDOW);

drawgraph_spiruline();

DrawScale(2);

while (!kbhit());

_setvideomode(-1);

}
```

Help.c

```
/*********************************************************************

NAME            HELP.C

AUTHOR          Pedro Pons

DESCRIPTION     Help for the keyboard commands

UPDATES
        20-09-95

*********************************************************************/
#include "help.h"

/* ==============================
    First aproximation to a help system
=============================== */


void help (void)
{
    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();


    sprintf(buffer,"*** MELISSA - Help ***");
    display_result(buffer,28,1);

    sprintf(buffer,"F1 ... Help.");
    display_result(buffer,10,3);
    sprintf(buffer,"ALT+F1 ... Repaint screen.");
    display_result(buffer,10,4);
    sprintf(buffer,"F2 ... System status.");
    display_result(buffer,10,5);
    sprintf(buffer,"F3 ... File management menu.");
    display_result(buffer,10,6);
    sprintf(buffer,"F5 ... Spirulina reactor.");
    display_result(buffer,10,7);
    sprintf(buffer,"ALT+F5 ... Spirulina reactor evolution.");
    display_result(buffer,10,8);
    sprintf(buffer,"CTR+F5 ... Spirulina reactor interval.");
    display_result(buffer,10,9);
    sprintf(buffer,"F6 ... Nitrifying reactor.");
    display_result(buffer,10,10);
    sprintf(buffer,"F7 ... Reactor 3.");
    display_result(buffer,10,11);
    sprintf(buffer,"F8 ... Reactor 4.");
    display_result(buffer,10,12);
    sprintf(buffer,"F9 ... Compounds simulated results.");
    display_result(buffer,10,13);
    sprintf(buffer,"ALT+F9 ... Concentration simulated results.");
    display_result(buffer,10,14);
    sprintf(buffer,"CTR+F9 ... Global simulation results.");
    display_result(buffer,10,15);
    sprintf(buffer,"F10 ... Alarms ON/OFF");
    display_result(buffer,10,16);
    sprintf(buffer,"CTR+F10 ... Run simulation.");
    display_result(buffer,10,17);
    sprintf(buffer,"Esc .. Main screen.");
    display_result(buffer,10,20);
}

#include "modspiru.c"
```

MelFct.c

```
/*********************************************************************
      NAME              MELFCT.C

      AUTHOR            Pedro L. Pons

      DESCRIPTION       Time related functions.

      UPDATES
            10-03-93
*********************************************************************/
#include "melfct.h"

unsigned long main_lastsamp = 0; /* last sampling for reactor 4*/
unsigned main_tsamp = TSAMP_MAIN;    /* sampling interval for reactor
4 */
unsigned long main_last_display;

/*---------------------------------------
      Timer pel control del bucle principal. DONA PEGUES  !!!
--------------------------------------*/

timebase_main(void)
      {
      time_t  ltime;
      char    buffer[100];
      struct  tm  *dt;
      double  minute;

      time(&ltime);
    if(main_last_display <ltime)
            {
            dt=localtime(&ltime);
            dt->tm_mon++;
            sprintf(buffer," %02d/%02d/%02d   %02d:%02d:%02d",dt-
>tm_mday,
            dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt->tm_sec);
            display_time(buffer);
        main_last_display=ltime;
            }

      time(&ltime);
    if (main_lastsamp==0)
            {
            display_status("timebase synchronisation ...");
        minute= TSAMP_MAIN / 60;
            while((ceil(dt->tm_min/minute)!=dt->tm_min/minute)||(dt-
>tm_sec!=0))
                  {
                  time(&ltime);
                  dt=localtime(&ltime);
                  dt->tm_mon++;
            if(main_last_display<ltime)
                        {
                        sprintf(buffer," %02d/%02d/%02d
%02d:%02d:%02d",dt->tm_mday,
                        dt->tm_mon,dt->tm_year,dt->tm_hour,dt-
>tm_min,dt->tm_sec);
                        display_time(buffer);
                  main_last_display=ltime;
                          }
                      }
          main_lastsamp=ltime;
              display_status(" ");
              return(0);
              }
      if ((main_lastsamp+main_tsamp)<=ltime)
              {
          main_lastsamp=main_lastsamp+main_tsamp;
              return(0);
              }
        return(ltime);
        }

/*--------------------------------------
        wait i seconds
--------------------------------------*/

#ifndef ADERSA
wait_time(int i)
#else
wait_time( i)
int i;
#endif
        {
        char    buffer[100];
        unsigned long start_time;

        while(timebase_main()==0)
              {
              }
        start_time=timebase_main();
        while(timebase_main()<start_time+i)
              {
              }
        }
```

Melissa.c

```
/*********************************************************************

NAME            MELISSA.C (ex. SPIRULIN.C)

AUTHOR          BINOIS C      (modified by FULGET N. ADERSA & PEDRO
PONS)

DESCRIPTION
      MAIN PROGRAM listing file

UPDATES
    20-09-95, 11-05-97

*********************************************************************/

#ifndef ADERSA
#include <graph.h>
#include <process.h>
#endif
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <dos.h>
#include <direct.h>
#include <io.h>
#include <conio.h>
#include <bios.h>
#include <math.h>

#include "userdef.h"
#include "ctrlspir.h"
#include "ctrlnitr.h"
#include "help.h"
#include "melissa.h"
#include "melfct.h"
#include "screen.h"
#include "pargene.h"
#include "simfile.h"

 /* #define _debugging_vars */

extern double average_var (VARS*,int);
extern datagraph spdata, actdata, ntdata;
unsigned long checkfloppy (void);

#define  __ALL_REACTORS
int      my_interrupt();
GPS_FILE *gps, *open_file_gps(), *activ_grp_gps();

int diskfull=0, diskwarning=0;  /* Controle the capacity of the disk
*/

int min_spir,       /* Count the minutes left to take a sample   */
    min_lique,      /* for each compartment                      */
    min_rhodo,
    min_nitri,
    min_glob;

int SAMP_SPIRU,      /* Define periods for taking samples */
    SAMP_RHODO,
    SAMP_LIQUE,
    SAMP_NITRI,
    SAMP_GLOB;

int pointer;

VARS    essai;
void alarms (void);
int control_process (ScreenViews active_reactor);
void keybdrv (ScreenViews *react, char chr);
void disk_storage (int input);
void parameters_adquisition(void);
void set_defaults(void);
void init_samples (void);
void check_disk (void);
void move_files( char*, char*);
void remove_file (char *name);

/*#define _testing_screens*/
#ifndef _testing_screens

main()
{
    char    chr;
    ScreenViews active_reactor=principal;  /* Indicates the info to
show */
    FILE *fout_spirulina;

/*---------------------------------------
    screen and graphical initialisation
-------------------------------------*/
    InitializeGraphicalStructures ();

#ifndef ADERSA
      screen_init();
#endif

/* ====================================
      log file initalisation
  ==================================== */

    check_disk();

      init_log_file();

/*---------------------------------------
      set interruption
-------------------------------------*/

      if( signal(SIGINT,my_interrupt) == (int(*)())-1)
            {
            _outtext("\nCouldn't set SIGINT *** Program Terminated
***\n");
            exit(0);
            }


/*---------------------------------------
      timebase synchronisation
-------------------------------------*/

      wait_time(1);

/*---------------------------------------
      open groups and active one
```

```
--------------------------------------*/

    gps=open_file_gps();
      gps=activ_grp_gps();
    set_gps(gps);
/*--------------------------------------
      variables and output file initialisation
--------------------------------------*/
      parameters_adquisition();
      init_vars();
      init_alarm();
    result(&active_reactor,0);
    init_samples();

/*--------------------------------------
      waiting loop
--------------------------------------*/

      do
      {
      if (!timebase_main())           /* Timer principal */
            {
            int in;

            check_disk();
#ifndef _debugging_vars
            check_network();             /* The check is going to
                                                            be made
once a minute */
            alarms();
#endif
            in=control_process(active_reactor);
            disk_storage(in);
            }
      if(kbhit())
            {
            chr=getch();
            keybdrv(&active_reactor, chr);
            }
      }
      while(1);
}
#else

main()
{

screen_init();
while (!kbhit());
_setvideomode (-1);
}

#endif
/*--------------------------------------
      interruption of programm
--------------------------------------*/

int     my_interrupt()
      {
      char    ch;
      signal(SIGINT,SIG_IGN);
      _clearscreen (_GWINDOW);
      _outtext("Terminate processing ? ");
      ch=getch();
      if((ch=='y')||(ch=='Y'))
            {
         close_grp_gps();
            _outtext("\nbye.... *** Program Terminated by user ***\n");
        wait_time(5);
#ifndef ADERSA
            _setvideomode(_DEFAULTMODE);
#endif
            exit(0);
            }
      if( signal(SIGINT,my_interrupt) == (int(*)())-1)
            {
            _outtext("\nCouldn't set SIGINT *** Program Terminated
***\n");
            exit(0);
            }
      _outtext("Continue...\n");
      return;
      }

/* =================================
    Keyboard driver routine.
================================== */


void keybdrv (ScreenViews *react, char chr)
{
int modified=0;
int new=1;

if (!chr)   /* Tecles de funcio */
    {
    chr = getch();
    switch (*react)
    {
    case status:        /* F10 turn on/off alarms */
      {
        if (chr == 0x44) {activated_alarms = (activated_alarms ?
0:1);
                    ++modified;
                    }
        break;
      }
      case simulationParams :
            {
            if (chr== 0x3f) {                                   /* F5
*/
                  InputConc(0);
                  ++modified;
                  }
            if (chr== 0x40) {                                   /* F6
*/
                  InputConc(1);
                  ++modified;
                  }
            if (chr== 0x41) {                                   /* F7
*/
                  InputStep (1);
                  ++modified;
                  }
            if (chr== 0x42) {                                   /* F8
*/
                  InputStep (0);
                  ++modified;
```

```
                    }
              }
              break;

      case filemanagement:          /* File manageing submenu */
        {
        if (chr== 0x3b) { *react= help_1;           /* F1 */
                ++modified;
              }
        if (chr== 0x3f) { move_disk();               /* F5 */
                      ++modified;
              }
        if (chr== 0x40) { move_files ("log.txt", "A:log.txt"); /* F6 */
              }
        if (chr== 0x41) { remove_file ("log.txt");      /* F7 */
                    }
        if (chr== 0x42) { directory ();       /* F8 */
                     *react = principal;
                    }
        if (chr== 0x43) { copy_disk();                /* F9 */
                      ++modified;
              }

        } break;
      default:               /* Main menu */
        {
        if (chr== 0x3b) { *react= help_1;           /* F1 */
                ++modified;
              }
        if (chr== 0x3c) { *react= status;           /* F2 */
                ++modified;
              }
        if (chr== 0x3d) { *react= filemanagement;           /* F3 */
                ++modified;
              }
        if (chr== 0x3f) { *react = spirulina;       /* F5 */
                ++modified;
              }
        if (chr== 0x6c) { *react = gspirulina;       /* Alt+F5 */
                      ++modified;
                    }
        if (chr== 0x62) { *react = igspirulina;       /* Crt+F5 */
                      ++modified;
                    }
        if (chr== 0x40) { *react = nitrifying;       /* F6 */
                ++modified;
              }
        if (chr== 0x6d) { *react = gnitrifying;       /* Alt+F6 */
                      ++modified;
                    }
        if (chr== 0x63) { *react = ignitrifying;       /* Crt+F6 */
                      ++modified;
                    }
        if (chr== 0x41) { *react = reactor_3;       /* F7 */
                ++modified;
              }
        if (chr== 0x42) { *react = reactor_4;       /* F8 */
                ++modified;
              }
        if (chr== 0x43) {
              *react = simulation1;       /* F9 */
              ++modified;
              }
        if (chr== 0x70) {
            *react = simulation2;        /* Alt+F9 */
            ++modified;
            }
      if (chr== 0x71) {
            *react = simulationParams;        /* Alt+F10 */
            ++modified;
            }
      if (chr== 0x66) {
            *react = simulation3;        /* Ctr-F9 */
            ++modified;
            }
      if (chr== 0x67) {                            /* F11 */
            /* The execution of the simulation must be performed */
            double H;

            H = tsim / sim_dt;              /* Number of steps = hours
/ sampling per. */
                                    /* Here we have to call to the
simulation routines. */
            display_status ("Simuling ...");
            modspiru2 (H, FCr0, FFRH, FdilH, FCiH, FCrH, FmasbioH,
FchonspH);
            PrintRes(H);
            display_status ("Simulation finished \a");
            display_status ("");
            ++modified;
            }
      if (chr== 0x68) {                               /* ALT+F1 */
            screen_init();
            ++modified;
            }
      }
      break;
      }   /* End of switch */
    } /* End of if keypressed = F* */
    else
    {
    if (chr == 27) { *react = principal;
                 ++modified;
                 }
         else
         {
         switch (*react)
            {
            case gspirulina :
            case igspirulina:
            case gnitrifying:
            case ignitrifying:
            case simulation1:
            case simulation2:
            case simulation3:
                  {
                  switch (chr)
                      {
                      case '+' : AdvanceScale ();
                                 break;
                      case '-' : GoBackScale();
                                 break;
                      case '*' : IncrementScale ();
                                 new=0;
                                 ++modified;
                                 break;
                      case '/' : DecrementScale();
                                 new=0;
```

```
                                ++modified;
                                break;
                        default : break;
                        }
                break;
                }


          }
       }        /* End of else */
    }

if (modified) { if (!result(react,new)) {result(react, new); }}
}


/* ======================================
      Local Loop
====================================== */

int control_process (ScreenViews active_reactor)
{
int out=0;
char buffer[100];


if ((--min_glob)==0)
    {
          /* Should call to global control */
    min_glob = SAMP_GLOB;
    }

if((--min_lique)==0)
      {
      if (control_lique(active_reactor)) out = out | 0x02;
    min_lique = SAMP_LIQUE;
      }

if((--min_rhodo)==0)
    {
    if (control_rhodo(active_reactor)) out = out | 0x08;
    min_rhodo = SAMP_RHODO;
    }

if((--min_nitri)==0)
      {
      if (control_nitrogen(active_reactor)) out = out | 0x04;
    min_nitri = SAMP_NITRI;
      }

if((--min_spir)==0)
    {
    if (control_spiru(active_reactor)) out = out | 0x01;
    min_spir = SAMP_SPIRU;
    }

_clearscreen(_GWINDOW);

if (out & 0x01 && active_reactor == gspirulina) result
(&active_reactor,0);
if (active_reactor != gspirulina && active_reactor != igspirulina &&
    active_reactor != simulation1 && active_reactor != simulation2 &&
    active_reactor != simulation3 )
    result (&active_reactor,0);
    #ifndef __Monitoring
    else
          next_control ();
    #endif

return out;
}


void alarms (void)
{
char buffer[100];

alarm_spiru();
alarm_lique();
alarm_nitri();
alarm_rhodo();

sprintf (buffer, "Accessing reactor alarms ...\n");
_outtext (buffer);
wait_time (1);

_clearscreen(_GWINDOW);
}


void disk_storage (int input)
{
FILE *fp;
double v1, v2;
char buffer[400], file[40];
time_t ltime;
struct tm *dt;

if (diskfull) return;

if (input & 0x01)       /* Spirulina output */
      {
    strcpy (file,"sp");
    time(&ltime);
    dt=localtime(&ltime);
    dt->tm_mon++;
    sprintf(file,"/gpsdat/sp%02d%02d%02d.out",dt->tm_year,dt-
>tm_mon,dt->tm_mday);

    fp = fopen (file, "rt");
    if (fp==NULL) { /* If can't open file -> new file */
                char buffer[400];

                fp = fopen (file, "at");  /* write heading */
                sprintf (buffer, "cxa_moy, nit_moy, production.sp,
react.temp, cons_prod_real.sp,  qe_real.sp, qe_real.value, Fr.sp,
Eb.sp\n");
                fwrite(buffer, 1, strlen(buffer), fp);

            }
      else    /* .... if could open, go to last position */
          {
          fclose (fp);
          fp = fopen (file, "at");
          }

    if (fp==NULL) { display_error ("Unable to open output file");
                return; }
```

```
     v1 = average_var(&cxa, 10);
     v2 = average_var(&nitrate, 10);
     wait_time(2);
     sprintf (buffer, "%02d:%02d %f %f %f %f %f %f %f %f %f\n",dt-
>tm_hour,dt->tm_min,
       v1 , v2 , production.sp, temperature.value,
       cons_prod_real.sp,  qe_real.sp, qe_real.value, Fr.value,
Eb.value);
       fwrite (buffer, 1, strlen(buffer), fp);
       fclose (fp);
       }

if (input & 0x04)       /* Nitrifying output */
       {
     strcpy (file,"sp");
     time(&ltime);
     dt=localtime(&ltime);
     dt->tm_mon++;
     sprintf(file,"/gpsdat/nt%02d%02d%02d.out",dt->tm_year,dt-
>tm_mon,dt->tm_mday);

     fp = fopen (file, "rt");
     if (fp==NULL) { /* if can't open -> new file */
                 char buffer[400];

                 fp = fopen (file, "at");  /* write heading */
                 sprintf (buffer, "DO.value, pH.value,
Temperature.value, Temperature.sp , Pressure.value,
AmmoniumConc.value, NitrateConc.value, AmmoniumConc.sp,
NitrateConc.sp\n");
                 fwrite(buffer, 1, strlen(buffer), fp);

              }
       else    /* .... if could open, I go to last position */
           {
           fclose (fp);
           fp = fopen (file, "at");
           }

     if (fp==NULL) { display_error ("Unable to open output file");
                 return; }
     v1 = average_var(&cxa, 10);
     v2 = average_var(&nitrate, 10);
     sprintf (buffer, "%02d:%02d %f %f %f %f %f %f %f %f %f\n",dt-
>tm_hour,dt->tm_min,
       NitrDo.value , NitrpH.value , NitrTemp.value, NitrTemp.sp,
       NitrPressure.value,  NitrAmCon.value, NitrNitrCon.value,
NitrAmCon.sp, NitrNitrCon.sp);
       fwrite (buffer, 1, strlen(buffer), fp);
       fclose (fp);
     }

}


void parameters_adquisition(void)
{
FILE *fp;
char input[100];
int ok=0;
float inputval=0;
long inputlong;
```

```
set_defaults();
fp=fopen(CONFIG_FILE, "rt");
if (fp==NULL) { /* Set defaults values */
            tech_log_file (" Warning ... no configuration file.");
            return;
            }.

while (!feof (fp))
      {
      ok=0;
      fscanf (fp, "%99s", input);
      if (strcmp(input, ""))
            {
            char *uperinput;
            uperinput = strupr(input);
            if (!strcmp (uperinput, "FILTERCXA")) {
                  fscanf (fp, "%f", &inputval);
                  ALPHAFILTERcxa = inputval;
                  ok++;
                  }
            if (!strcmp (uperinput, "FILTERNITRATE")) {
                  fscanf (fp, "%f", &inputval);
                  ALPHAFILTERnitrate = inputval;
                  ok++;
                  }
            if (!strcmp (uperinput, "FILTEREB")) {
                  fscanf (fp, "%f", &inputval);
                  ALPHAFILTEREb = inputval;
                  ok++;
                  }
            if (!strcmp (uperinput, "FILTERPH")) {
                  fscanf (fp, "%f", &inputval);
                  ALPHAFILTERpH = inputval;
                  ok++;
                  }
            if (!strcmp (uperinput, "SAMPSPIRU")) {
                fscanf (fp, "%i", &inputlong);
                SAMP_SPIRU = inputlong;
                ok++;
                }
            if (!strcmp (uperinput, "SAMPRHODO")) {
                fscanf (fp, "%i", &inputlong);
                SAMP_RHODO= inputlong;
                ok++;
                }
            if (!strcmp (uperinput, "SAMPLIQUE")) {
                fscanf (fp, "%i", &inputlong);
                SAMP_LIQUE = inputlong;
                ok++;
                }
            if (!strcmp (uperinput, "SAMPNITRI")) {
                fscanf (fp, "%i", &inputlong);
                SAMP_NITRI = inputlong;
                ok++;
                }
            if (!strcmp (uperinput, "SAMPGLOB")) {
                fscanf (fp, "%i", &inputlong);
                SAMP_GLOB = inputlong;
                ok++;
                }
      if (!ok) { /* Error reading parameter file, keyword not valid */
                                /* Set default values */
                 char buffer[200];
                 set_defaults();
```

```
                                /* Error message */
                                sprintf (buffer, "Unknown parameter : %s",input);
                                display_error(buffer);
                                _clearscreen(_GWINDOW);
                                }
                        }
                }
        }

/*  =========================
        Set all the varibles that can be parametrized
        to a default value.
========================= */


void set_defaults(void)
{
ALPHAFILTERcxa = 1;
ALPHAFILTERnitrate = 1;
ALPHAFILTEREb = 1;
ALPHAFILTERpH = 1;
SAMP_SPIRU = 1;
SAMP_LIQUE = 1;
SAMP_RHODO = 1;
SAMP_NITRI = 1;
SAMP_GLOB = 1;
}

void init_samples (void)
{
min_spir = SAMP_SPIRU;
min_lique = SAMP_LIQUE;
min_rhodo = SAMP_RHODO;
min_nitri = SAMP_NITRI;
min_glob = SAMP_GLOB;
}

void check_disk (void)
{
int disk;
struct diskfree_t df;
unsigned long lliure;
char buffer[100];

display_status ("Checking disk ... ");

_dos_getdiskfree(0, &df);    /* Ud per defecte */
lliure = (unsigned long) df.avail_clusters * (unsigned long)
df.sectors_per_cluster * (unsigned long) df.bytes_per_sector;

if (lliure < 100000) {diskfull=1; }


if (lliure < 400000) {
                /* Warning */
                display_error ("Disk nearly full !!!");
                diskwarning = 1;
                }
                else
                { diskwarning = 0;
                  diskfull = 0;
                }
if (lliure < 100000) {
```

```
                /* Disable disk saving */
                sprintf (buffer,"Disabling disk saving ...");
                display_error (buffer);
                return;
                }
return;
}


unsigned long checkfloppy (void)
{
struct diskinfo_t diskinfo;
struct diskfree_t diskfree;
int st;
unsigned long lliure;

diskinfo.drive = 0; /* ud A: */
diskinfo.head = 0;
diskinfo.track = 1;
diskinfo.sector = 2;
diskinfo.nsectors = 1;

st = _bios_disk (_DISK_VERIFY, &diskinfo);
if (0x8000 & st) return -1;

_dos_getdiskfree(1, &diskfree);

lliure = (unsigned long) diskfree.avail_clusters *
       (unsigned long)  diskfree.sectors_per_cluster *
       (unsigned long) diskfree.bytes_per_sector;


return lliure;
}

int copia (char *or, char *des, unsigned long lliure)
{
FILE *o, *d;
char c;

o = fopen (or,"rb");
if (o==NULL)  {
            display_status ("Error : source file does not exist.");
            printf("\a\a");
            wait_time(2);
            display_status (" ");
            return -1;
            }
fseek(o, 0L, SEEK_END);
if (lliure <= ftell(o)) {
                  display_status ("Not enough space in disk :");
                  printf ("\a\a");
                  wait_time(2);
                  display_status (" ");
                  return -5;
                  }
fseek (o,0L,SEEK_SET);

d = fopen (des, "wb");
if (d==NULL)  {
            display_status ("Error opening destination file.");
            printf("\a\a");
            wait_time(2);
            display_status (" ");
            return -2;
            }

fread(&c, 1,sizeof(char), o);
do  {
    fwrite(&c, 1, sizeof(char), d);
    fread (&c, 1, sizeof(char), o);
    } while (!feof(o) && !ferror(o) && !ferror(d) );

if (ferror(o)) {
            display_status ("Error during the copy process in source
file.");
            printf("\a\a");
            wait_time(2);
            display_status (" ");
            return -3;
            }
if (ferror(d)) {
             display_status ("Error during the copy process in
destination file.");
             printf("\a\a");
             wait_time(2);
             display_status (" ");
             return -4;
             }
fclose (d);
fclose (o);
return 0;
}


void move_files (char *name, char *destination)
{
unsigned long lliure;

/* Verify the floppy drive and free space */
if (strcmp (destination, "NUL") && strcmp(destination, "nul") )
    {
    lliure = checkfloppy();
    if (lliure == -1) {
                display_status ("Drive A: is not ready.");
                printf ("\a\a");
                wait_time(2);
                display_status (" ");
                return;
                }
    }

if (copia(name, destination, lliure)==0)
      {   /* If ok => remove the original */
      if (remove(name)==-1)
            {
            sprintf (buffer,"Could not delete the file %s",name);
            display_status(buffer);
            printf("\a\a");
            display_status (" ");
            wait_time(2);
            }

      }

}

void remove_file (char *name)
```

```
{
char c;

display_status ("Are you sure ? (Y/N)");
printf ("\a\a");

while (!kbhit());
c = getch();
if (c!= 'y' && c!= 'Y')
            {
            display_status(" ");
            return;
            }


if (remove(name)==-1)
      {
      sprintf (buffer,"Could not delete the file %s",name);
      display_status(buffer);
      printf("\a\a");
      display_status (" ");
      wait_time(2);
      }
display_status(" ");
}

/* Due to the DOS version, no more files can be passed as parameters
to the linker,
so here will be included the spirulina simulator.
*/
/*#include "modspiru.c"*/
/* This file should be included here, but due to the size of the
present file
it will be included in help.c
*/
```

ModSpiru.c

```
/*
modspiru.c

Subroutine for the estimation of the quality of the biomass
of the photoautotroph (spirulina) compartment
based on the first principles model built by LGCB (TN 19.1 and
19.2)

J.J. Leclercq
ADERSA
April 1997

-----------------------------------------------------------------
-------

Synopsis:
=========
modspiru(H, Cr0, FRH, dilH, CiH, CrH, fmasbioH, chonspH)
Input arguments :
-----------------
H : length of simulation (expressed in number of sampling period
dt)
Cr0 : initial concentrations in the reactor (kg/m3)
FRH : time variation of the incident radiant energy flux (W/m2)
dilH : time variation of the dilution (1/h)
CiH : time variation of the concentrations in the incoming flow
(kg/m3)
Ouput arguments :
-----------------
CrH : time variation of the concentrations in the reactor
(kg/m3)
fmasbioH : time var. of the mass fraction of the biomass
(dimensionless)
chonspH : time var. of the global formula of the biomass
(dimensionless)

Storage in Cr0, CiH and CrH (at a given moment) :
      (1) : total biomass
      (2) : active biomass
      (3) : chorophyll
      (4) : phycocyanins
      (5) : proteins
      (6) : nitrate
      (7) : sulfate
      (8) : vegetative biomass
      (9) : exopolysaccharide

Storage in fmasbioH (at a given moment) :
      (1) : phycocyanins
      (2) : other proteins
      (3) : chorophylls
      (4) : biomass
      (5) : glycogen
      (6) : exopolysaccharide

Storage in chonspH (at a given moment)
of the global formula C Ha Ob Nc Sd Pe :
      chonspH(1) = a
      chonspH(2) = b
      chonspH(3) = c
      chonspH(4) = d
```

```
            chonspH(5) = e
      */

#include "parmodel.h"
#include "pargene.h"
#include "simfile.h"
#include "math.h"
#include "screen.h"
#include <stdio.h>
#include <graph.h>

extern void pro_auto();
extern void forglob();
extern void compo();



void  modspiru2(H, Cr0, FRH, dilH, CiH, CrHout, fmasbioH, chonspH)
double H;
char *Cr0, *FRH, *dilH, *CiH,*CrHout, *fmasbioH, *chonspH;
{
      short i, j;
    double FR, dil, Ci[nsig+1], Cr[nsig+1], CrLoc[nsig+1];
      double chonsp[ncoef+1], fmasbio[ncomp+1];
    double aux;
    FILE *fCr0, *fFRH, *fdilH, *fCiH,
       *fCrHout, *ffmasbioH, *fchonspH;

    fCr0 = fopen (Cr0, "r");
    fFRH = fopen (FRH, "r");
    fdilH = fopen (dilH, "r");
    fCiH = fopen (CiH, "r");
    fCrHout = fopen (CrHout, "w");
    ffmasbioH = fopen (fmasbioH, "w");
    fchonspH = fopen (chonspH, "w");

    if ( fCr0 == NULL || fFRH == NULL ||
       fdilH == NULL || fCiH == NULL ||
       fCrHout == NULL ||
       ffmasbioH == NULL || fchonspH == NULL )
      { display_error ("\n Error reading parameter files for the
simulation.\a\n");
        return;
      } else {
      display_status ("Simuling ...");
      }


    for (j=1; j<=nsig; j++) {
      /*       CrH[0][j] = Cr0[j];                */
      fscanf (fCr0, "%lf", &aux);
      CrLoc[j] = aux;
      fprintf (fCrHout, "%lf ", aux);
      }
    fprintf (fCrHout, "\n");
      for (i=0; i<=H; i++)
      {
      /*           FR = FRH[i];                  */
      fscanf (fFRH, "%lf", &FR);
      /*           dil = dilH[i];                */
      fscanf (fdilH, "%lf", &dil);
      for (j=1; j<=nsig; j++) {
          /*             Ci[j] = CiH[i][j];        */
          fscanf (fCiH, "%lf", &Ci[j]);
```

```
            /*              Cr[j] = CrH[i][j];          */
            Cr[j] = CrLoc[j];
            }

              /* global formula of the biomass (CHONSP) */
        forglob(Cr, chonsp);
        for (j=1; j<=ncoef; j++) {
        /*              chonspH[i][j] = chonsp[j];          */
            fprintf (fchonspH, " %lf ", chonsp[j]);
            }
        fprintf (fchonspH, "\n");

              /* mass fraction of the biomass (dimensionless) */
        compo(Cr, fmasbio);
        for (j=1; j<=ncomp; j++) {
        /*              fmasbioH[i][j] = fmasbio[j];        */
            fprintf (ffmasbioH, " %lf ", fmasbio[j]);
            }
        fprintf (ffmasbioH, "\n");

        tech_log_file ("Entrant.");

              /* concentration (in kg/m3) of each compound in the reactor
*/
        pro_auto(FR, dil, Ci, Cr);
        tech_log_file ("Sortint");
        for (j=1; j<=nsig; j++) {
        /*              CrH[i+1][j] = Cr[j];        */
            fprintf (fCrHout, " %lf ", Cr[j]);
            CrLoc[j] = Cr[j];
            }

        fprintf (fCrHout, "\n");
        }
      fclose (fCr0);
      fclose (fFRH);
      fclose (fdilH);
      fclose (fCiH);
      fclose (fCrHout);
      fclose (ffmasbioH);
      fclose (fchonspH);

}

void  compo(Cr, fmasbio)
double Cr[nsig+1], fmasbio[ncomp+1];
{
        fmasbio[1] = Cr[4] / Cr[1];
        fmasbio[2] = (Cr[5] - Cr[4]) / Cr[1];
        fmasbio[3] = Cr[3] / Cr[1];
        fmasbio[4] = (Cr[2] - Cr[3] - Cr[5]) / Cr[1];
        fmasbio[5] = (Cr[8] - Cr[2]) / Cr[1];
        fmasbio[6] = Cr[9] / Cr[1];
}

void  forglob(Cr, chonsp)
double Cr[nsig+1], chonsp[ncoef+1];
{
        double abm, glym, epsm, abn, glyn, epsn, abf, glyf, epsf, ntot;

        /* mass fraction of active biomass */
        abm = Cr[2] / Cr[1];
        /* mass fraction of glycogen */
        glym = (Cr[8] - Cr[2]) / Cr[1];
      /* mass fraction of exopolysaccharide */
      epsm = Cr[9] / Cr[1];

      /* molar ratio of active biomass */
      abn = abm / Mab;
      /* molar ratio of glycogen */
      glyn = glym / Mgly;
      /* molar ratio of exopolysaccharide */
      epsn = epsm / Meps;

      ntot = abn + glyn + epsn;
      /* molar fraction of active biomass */
      abf = abn / ntot;
      /* molar fraction of glycogen */
      glyf = glyn / ntot;
      /* molar fraction of exopolysaccharide */
      epsf = epsn / ntot;

      /* hydrogen coefficient */
      chonsp[1] = 1.566*abf + 1.67*glyf + 1.65*epsf;
      /* oxygen coefficient */
      chonsp[2] = .405*abf + .711*glyf + .95*epsf;
      /* nitrogen coefficient */
      chonsp[3] = .192*abf;
      /* sulphure coefficient */
      chonsp[4] = .0052*abf + .0007*glyf + .015*epsf;
      /* phophorus coefficient */
      chonsp[5] = .0063*abf;
}

extern void rx_auto();
void pro_auto(FR, dil, Ci, Cr)
double FR, dil, Ci[nsig+1], Cr[nsig+1];
{
      short i;
      double RXA, REPS;
      double xCCH, xCPC, xCN, xCS, xCXV;
      double aa, bb, cc, dd, ee;
      double ri[nsig+1], dCr[nsig+1];

    xCCH = Cr[3];
      xCPC = Cr[4];
      xCN  = Cr[6];
      xCS  = Cr[7];
      xCXV = Cr[8];

      aa = xCN / (KN + xCN);
      bb = xCS / (KS + xCS);
      cc = xCPC / (KPC + xCPC);
      dd = KN / (KN + xCN);
      ee = KS / (KS + xCS);

      /* calculation of RXA et REPS */
      rx_auto(xCPC,xCCH,xCXV,FR,&RXA,&REPS);

    /* calculation of the 9 mean volumic growth rates */
      ri[1] = RXA + REPS;                                                  /*
rXT  */
      ri[2] = RXA * aa * bb;                                               /*
rXA  */
      ri[3] = zCH * ri[2];                                                 /*
rCH  */
      ri[4] = zPC * RXA * (aa*bb - (dd+ee));                               /*
rPC  */
```

```
        ri[5] = zP * RXA * (aa*bb - qq*ee);                              /*
rP     */
        ri[6] = -YNXA * ri[2];                                           /*
rN     */
        ri[7] = -YSXA * ri[2] - YSEPS * REPS * aa * bb;                  /*
rS     */
        ri[8] = RXA * (aa*bb + cc*(dd+ee));                              /*
rXV    */
        ri[9] = REPS * aa * bb + (ri[1] - ri[8]) * (dd + ee);            /*
rEPS */

        /* calculation of derivatives and integration */
        /* integration step for Euler method  = dt */
        for (i=1; i<=nsig; i+=1)
        {
              dCr[i] = ri[i] + dil * (Ci[i] - Cr[i]);
              Cr[i] = Cr[i] + dCr[i] * sim_dt;
        }

}

void  rx_auto(CPC,CCH,CXV,FR,RXA,REPS)
double  CPC, CCH, CXV, FR, *RXA, *REPS;
{
        /* internal variables */
/*      double yy[nZ+1]; */
    double *yy;
        double a1, a2, alpha, delta, sXA, sEPS, pijz, z, REPS1, REPS2,
A, PE;
        double z0, kstep;
        short i;
yy = (double*) malloc (sizeof(double) * nZ+1);
if (yy == NULL) {
    printf ("ERROR IN MALLOC.\n\n");
    exit (-1);
    }

        a1 = Ea * (CPC+CCH);
        a2 = Ea * (CPC+CCH) + Es*CXV;
        alpha = sqrt(a1 / a2);
        delta = (sqrt(a1 * a2)) * RT;

        /*
        Computation of RXA et REPS1
        Integration interval : [z0, 1]
        This interval is divided into 'nstep' equal parts
        Integration : trapezium method
        */
        z0 = 1.e-5 / RT;
        kstep = (1. - z0) / nstep;
        i = 0;
        for (z=z0; z<=1.; z+=kstep)
            {
            i += 1;
            yy[i] =
2*FR/z*cosh(delta*z)/(cosh(delta)+alpha*sinh(delta));
            }
        sXA = 0.;
        sEPS = 0.;
        i = 0;
        for (z=z0; z<1.; z+=kstep)
            {
            i += 1;
            pijz = (yy[i] + yy[i+1]) / 2.;
```

```
          if (pijz>=Fmin)
             {
             sXA += z * pijz / (Kj+pijz);
             sEPS += z * pijz / (KjEPS+pijz);
             }
          }
      *RXA = 2. * mupM * CPC * sXA * wiv * kstep;

      REPS1 = 2. * mupMEPS * CPC * sEPS * wiv * kstep;
      A = 4*FR * alpha * sinh(delta) / RT / (cosh(delta) +
alpha*sinh(delta));
      PE = 1.222e-5 * A + 1.267;
      REPS2 = 29.33 * (2.874*PE - 3.568) * *RXA / 23.096 / (3.33-
1.92*PE);
      *REPS = (REPS1 + REPS2) / 2.;

free (yy);
}

/* This functions gets the resulting files and creates three final
files with the final results.
*/


void PrintRes (double H)
{
int i;
FILE *pf;
FILE *fCrH, *fFRH, *fdilH, *fmasbioH, *fchonspH;
double v1, v2, v3, v4, v5, v6, v7, v8, v9;
double frh, dilh;
    /*
    Saving results into 3 files *.res :
    1_ concentrations of the compounds in the reactor : conc.res
    2_ mass fraction of the biomass : compo.res
    3_ global formula (CHONSP) of the biomass : glob.res
    */


    fCrH = fopen (FCrH, "r");
    fFRH = fopen (FFRH, "r");
    fdilH = fopen (FdilH, "r");

    pf = fopen("/gpssim/conc.res", "w");
    for (i=0; i<=H; i++)
    {
    fscanf (fFRH, " %lf", &frh);
    fscanf (fdilH, " %lf", &dilh);
    fscanf (fCrH, "%lf %lf %lf %lf %lf %lf %lf %lf %lf", &v1, &v2,
&v3,
          &v4, &v5, &v6, &v7, &v8, &v9);
/* OJO. Ôs per treure columnes de dades. */
/*    fprintf(pf,"%10.2f %12.5e %12.5e %12.5e %12.5e %12.5e %12.5e
%12.5e %12.5e %12.5e %12.5e %12.5e\n",
      i*sim_dt,frh,dilh, v1, v2, v3, v4, v5, v6, v7, v8, v9);
*/

    fprintf(pf,"%12.5e %12.5e %12.5e %12.5e %12.5e %12.5e %12.5e
%12.5e %12.5e\n",
      v1, v2, v3, v4, v5, v6, v7, v8, v9);
    }
    fclose(pf);
    fclose (fCrH);
    fclose (fFRH);
```

```
     fclose (fdilH);

     fmasbioH = fopen (FmasbioH, "r");
     pf = fopen("/gpssim/compo.res", "w");
     for (i=0; i<=H; i++)
     {
     fscanf (fmasbioH, "%lf %lf %lf %lf %lf %lf", &v1, &v2, &v3,
           &v4, &v5, &v6);
     fprintf(pf,"%10.2f %12.5e %12.5e %12.5e %12.5e %12.5e
%12.5e\n",i*sim_dt,
       v1,v2,v3,v4, v5, v6);
     }
     fclose(pf);
     fclose (fmasbioH);

     fchonspH = fopen (FchonspH, "r");

     pf = fopen("/gpssim/glob.res", "w");
     for (i=0; i<=H; i++)
     {
     fscanf (fchonspH, "%lf %lf %lf %lf %lf", &v1, &v2, &v3,
           &v4, &v5);
     fprintf(pf,"%10.2f %12.5e %12.5e %12.5e %12.5e %12.5e\n",i*sim_dt,
       v1, v2, v3, v4, v5);
     }
     fclose(pf);
     fclose (fchonspH);
}


void ShowSimulationParams(void)
{
double cr0[10], cih[10];
double val0, val1, aux;
double dil0, dil1;
int change, dilchange;
int cr0na, cihna, frna, dilna;           /* Flags cr0 nOT aVAILABLE...
*/
FILE *fcr, *fci;
char bf[100];
int i;

cr0na=0; cihna=0; frna=0; dilna = 0;
change = 0; dilchange = 0;

use_display_window();
_clearscreen(_GWINDOW);
use_message_window();


fcr = fopen ( FCr0, "r");
if (fcr == NULL) {
      cr0na=1;
      } else {
      fscanf (fcr, "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
            &cr0[1], &cr0[2], &cr0[3], &cr0[4], &cr0[5]
            , &cr0[6], &cr0[7], &cr0[8], &cr0[9]);
      }
fclose (fcr);
fci = fopen ( FCiH, "r");
if (fci == NULL) {
      cihna=1;
      } else {
```

```
      fscanf (fci, "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
            &cih[1], &cih[2], &cih[3], &cih[4], &cih[5]
            , &cih[6], &cih[7], &cih[8], &cih[9]);
      }
fclose (fci);
fci = fopen ( FFRH, "r");
if (fci == NULL) {
      frna=1;
      } else {
      int finished = 0;
      fscanf (fci, "%lf", &val0);
    val1 = val0;
      while (!feof (fci) && !finished) {
            change++;
            fscanf (fci, "%lf", &aux);
            if (aux != val0) {
                  val1 = aux;
                  finished++;
                  }
            }
      }
fclose (fci);

fci = fopen ( FdilH, "r");
if (fci == NULL) {
      dilna=1;
      } else {
      int finished = 0;
      fscanf (fci, "%lf", &dil0);
    dil1 = dil0;
      while (!feof (fci) && !finished) {
            dilchange++;
            fscanf (fci, "%lf", &aux);
            if (aux != dil0) {
                  dil1 = aux;
                  finished++;
                  }
            }
      }
fclose (fci);
dilchange--; change--;

set_title ("Simulation parameters");
display_result (" [Initial    [incoming", 28, 2);
display_result ("in reactor]    flow]", 28, 3);
display_result ("Total biomass :",7 , 4);
display_result ("Active biomass :", 7 ,5);
display_result ("Chorophyll :", 7 ,6);
display_result ("Phycocyanins :", 7 ,7);
display_result ("Proteins :", 7 ,8);
display_result ("Nitrate :", 7 ,9);
display_result ("Sulfate :", 7 ,10);
display_result ("Vegetative biomass :", 7 ,11);
display_result ("Exopolysaccharide :", 7 ,12);

for (i=1; i <= 9; i++) {
      if (cr0na==0) {
            sprintf (bf, "%6.3lf", cr0[i]);
            } else {
            sprintf (bf, "Not Available ");
            }
      if (cihna==0) {
            sprintf (bf, "%s     %6.3lf ", bf, cih[i]);
            } else {
```

```
            sprintf (bf, "%s     Not Available ", bf);
            }
      display_result (bf, 30, 3+i);
      }
if (frna == 0) {
      sprintf (bf, "Flux step          %6.2lf at %6.1f => %6.2lf",
val0, change * sim_dt, val1);
      } else {
      sprintf (bf, "Flux step not available.");
      }
display_result (bf, 7, 14);
if (dilna == 0) {
      sprintf (bf, "Dilution rate step %6.2lf at %6.1f => %6.2lf",
dil0, dilchange * sim_dt, dil1);
      } else {
      sprintf (bf, "Dilution rate not available.");
      }
display_result (bf, 7, 15);
display_result ("    F5          F6        F7       F8", 7, 17);
display_result ("[initial in  [incoming  Flux  Dilution", 7, 18);
display_result ("  reactor]     flow]    step  rate step", 7, 19);
}


int InputStep (int flux)
{
float val0, val1;
float stepTime;
int i, i2, H;
FILE *fp;

use_display_window();
_clearscreen(_GWINDOW);
use_message_window();

if (flux == 1) {
      center_display_result ("Enter new flux step.",3);
      } else {
      center_display_result ("Enter new dilution rate step.",3);
      }

display_result ("Initial value : ", 7, 10);
display_result ("Time step (h) : ", 7, 11);
display_result ("Final value   : ", 7, 12);

GetFloatXY(27,10, &val0);
GetFloatXY(27,11, &stepTime);
GetFloatXY(27,12, &val1);

if (flux == 1) {
      fp = fopen ( FFRH , "w");
      } else {
      fp = fopen ( FdilH, "w");
      }
H = tsim / sim_dt;
i2 = (int) stepTime / sim_dt;
for (i = 0; i <= i2; i++)
      {
      fprintf (fp, "%f\n", val0);
      }
for (i = i2+1; i <= H; i++)
      {
      fprintf (fp, "%f\n", val1);
      }
```

```
fclose (fp);
return 1;
}



int InputConc (int flux)
{
float val[10];
int x, H, i;
FILE *fp;

if (flux == 1) {
      x = 42;
      center_display_result ("Enter incoming flow concentrations", 1);
      } else {
      x = 31;
      center_display_result ("Enter initial concentrations in
reactor", 1);
      }
display_result (" [Initial    [incoming", 28, 2);
display_result ("in reactor]    flow]", 28, 3);
display_result ("Total biomass :",7 , 4);
display_result ("Active biomass :", 7 ,5);
display_result ("Chorophyll :", 7 ,6);
display_result ("Phycocyanins :", 7 ,7);
display_result ("Proteins :", 7 ,8);
display_result ("Nitrate :", 7 ,9);
display_result ("Sulfate :", 7 ,10);
display_result ("Vegetative biomass :", 7 ,11);
display_result ("Exopolysaccharide :", 7 ,12);

GetFloatXY(x, 4, &val[1]);
GetFloatXY(x, 5, &val[2]);
GetFloatXY(x, 6, &val[3]);
GetFloatXY(x, 7, &val[4]);
GetFloatXY(x, 8, &val[5]);
GetFloatXY(x, 9, &val[6]);
GetFloatXY(x, 10, &val[7]);
GetFloatXY(x, 11, &val[8]);
GetFloatXY(x, 12, &val[9]);

H = (int) tsim / sim_dt;
if (flux == 1) {
      fp = fopen (FCiH, "w");
      if (fp == NULL) {
            display_status ("ERROR : Could not open file.");
            wait_time (2);
            return 0;
            }
      for (i=0; i <= H; i++) {
            fprintf (fp, "%lf %lf %lf %lf %lf %lf %lf %lf %lf\n",
(double) val[1], (double) val[2],
                        (double) val[3], (double) val[4], (double)
val[5], (double) val[6],
                        (double) val[7], (double) val[8], (double)
val[9]);
            }
      } else {
      fp = fopen (FCr0, "w");
      if (fp == NULL) {
            display_status ("ERROR : Could not open file.");
            wait_time (2);
```

```
                return 0;
                }
        for (i=1; i <= 9; i++) {
                fprintf (fp, "%lf \n", (double) val[i]);
                }
        }
fclose (fp);

return 1;
}


int GetFloatXY(int x, int y, float *f)
{
char bf[70];
char bf2[20];
int n;

bf[0] = '\0';
do {
        bf2[0] = '\0';
        display_result ("                ",x,y);
        use_display_window();
        _settextposition(y,x);
        scanf ("%s", bf);
        n = sscanf (bf, "%f%s", f, bf2);
        use_message_window();
        } while (n != 1 || bf2[0] != '\0');

return 1;
}


int GetIntXY(int x, int y, int *f)
{
char bf[70];
char bf2[20];
int n;

bf[0] = '\0';
do {
        bf2[0] = '\0';
        display_result ("                ",x,y);
        use_display_window();
        _settextposition(y,x);
        scanf ("%s", bf);
        n = sscanf (bf, "%i%s", f, bf2);
        use_message_window();
        } while (n != 1 || bf2[0] != '\0');

return 1;
}
```

Results.c

```
/*********************************************************************

NAME            RESULTS.C

AUTHOR          Pedro Pons

DESCRIPTION     Routines for displaying data, and managing log files.

UPDATES
        27-05-96

*********************************************************************/
#include "results.h"


#define HA 4
#define VA -2
#define THA -1
#define TVA -4

void move_disk (void);
void file_menu(void);
unsigned long checkfloppy (void);
void ShowSimulationParams(void);

/* This file contains all the functions that are in
    relation with the displaying of results     */

/* ===================================
        The tech-log is made to separate the technical warnings
related to
        the computers systems from the warnings and errors related to
        all the plant.
   =================================== */

void tech_log_file (char *buffer)
{
FILE *fp;
int lng;
char bf[200];
time_t ltime;
struct  tm  *dt;

if (diskfull) return;

fp = fopen ("techlog.txt", "at");
if (fp==NULL) { use_message_window();
                _outtext("Couldn't open log file.\n");
                use_display_window();
            }
            else {
                time(&ltime);
                dt=localtime(&ltime);
                dt->tm_mon++;
                sprintf(bf," %02d/%02d/%02d    %02d:%02d:%02d Msg :
",dt->tm_mday,
                dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt-
>tm_sec);

                strcat (bf, buffer);
                strcat(bf,"\n");
```

```
                    lng = strlen(bf);
                    fwrite (bf, 1, lng, fp);
                    fclose(fp);
                }
}


/* ===================================
    Routine to write a message in the log file
=================================== */

void log_file (char *buffer)
{
FILE *fp;
int lng;
char bf[200];
time_t ltime;
struct  tm  *dt;

if (diskfull) return;

fp = fopen ("log.txt", "at");
if (fp==NULL) { use_message_window();
                    _outtext("Couldn't open log file.\n");
                    use_display_window();
                }
                else {
                    time(&ltime);
                    dt=localtime(&ltime);
                    dt->tm_mon++;
                    sprintf(bf," %02d/%02d/%02d   %02d:%02d:%02d Msg :
",dt->tm_mday,
                    dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt-
>tm_sec);

                    strcat (bf, buffer);
                    strcat(bf,"\n");
                    lng = strlen(bf);
                    fwrite (bf, 1, lng, fp);
                    fclose(fp);
                }
}

void error_log_file (char *buffer)
{
FILE *fp;
int lng;
char bf[200];
time_t ltime;
struct  tm  *dt;

if (diskfull) return;

fp = fopen ("log.txt", "at");
if (fp==NULL) { use_message_window();
                    _outtext("Couldn't open log file.\n");
                    use_display_window();
                }
                else {
                    time(&ltime);
                    dt=localtime(&ltime);
                    dt->tm_mon++;
                    sprintf(bf," %02d/%02d/%02d   %02d:%02d:%02d Error :
",dt->tm_mday,
                  dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt-
>tm_sec);

                  strcat (bf, buffer);
                  strcat(bf,"\n");
                  lng = strlen(bf);
                  fwrite (bf, 1, lng, fp);
                  fclose(fp);
              }
}

/* ====================================
        General display results
==================================== */


int result(ScreenViews *display_reactor, int new)
{
#ifndef __Monitoring
        next_control ();
#endif

switch(*display_reactor)      /* Choose the info to show */
    {
    case help_1 : help();
                  break;
    case filemanagement : file_menu();
                  break;
    case principal : result_principal();
                     break;
    case spirulina :  result_spirulina();
                      break;
    case gspirulina : SelectSpirulineGraph ();
                      if (!drawgraph(new,0)){

*display_reactor=principal;
                                        return 0;
                                        }
                      break;
    case igspirulina : if (new) SelectSpirulineGraph ();
                      if (!drawgraph(new,1)) {

*display_reactor=principal;
                                         return 0;
                                         }
                      break;
    case nitrifying : result_nitrogen();
                     break;
    case gnitrifying : SelectNitrifyingGraph ();
                      if (!drawgraph(new,0)){

*display_reactor=principal;
                                        return 0;
                                        }
                      break;
    case ignitrifying : if (new) SelectNitrifyingGraph ();
                      if (!drawgraph(new,1)) {

*display_reactor=principal;
                                         return 0;
                                         }
                      break;
    case reactor_3 : result_reactor3();
                    break;
    case reactor_4 : result_reactor4();
                     break;
    case simulation1 : SelectSimulation1Graph();
                       if (!drawgraph(new,0)){

*display_reactor=principal;
                                            return 0;
                                            }
                     break;
    case simulation2 : SelectSimulation2Graph();
                       if (!drawgraph(new,0)){

*display_reactor=principal;
                                            return 0;
                                            }
                     break;
    case simulation3 : SelectSimulation3Graph();
                       if (!drawgraph(new,0)){

*display_reactor=principal;
                                            return 0;
                                            }
                     break;
    case status : result_status();
                  break;
      case simulationParams:
            {
            ShowSimulationParams();
            break;
            }
    default : break;
    }
return 1;
}


/*====================================
    Routine for spirulina displaying results
  ==================================== */

result_spirulina()
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

        set_title ("Spirulina compartment");

        use_display_window();
        _clearscreen(_GWINDOW);
        use_message_window();


        display_result ("Value", 27, 2);
        display_result ("SetPoint", 38, 2);

        _moveto (30, 100);
        _lineto (375,100);
        _moveto (190, 80);
        _lineto (190, 190);
        _moveto (280, 80);
```

```
 _lineto (280, 190);

 display_result("Biomass (mg/l)",7,4);
 display_result("Nitrate (mg/l)",7,5);
 sprintf(buffer,"%.1f",cxa.value);
 display_result(buffer,27,4);
 sprintf(buffer,"%.1f",nitrate.value);
 display_result(buffer,27,5);
 sprintf(buffer,"%.1f",cxa.sp);
 display_result(buffer,39,4);
 sprintf(buffer,"%.1f",nitrate.sp);
 display_result(buffer,39,5);

 display_result("Eb      (W/m2) ",7,6);
 sprintf(buffer,"%.1f", Eb.value);
 display_result(buffer,27,6);
 sprintf(buffer,"%.1f", Eb.sp);
 display_result(buffer,39,6);

 display_result("Fr      (W/m2) ",7,7);
 sprintf(buffer,"%.1f", Fr.value);
 display_result(buffer,27,7);
 sprintf(buffer,"%.1f", Fr.sp);
 display_result(buffer,39,7);


 display_result ("Setpt", 21,12);
 display_result ("Model", 29,12);
 display_result ("Realised",36,12);
 display_result ("Mesured", 45, 12);

 _moveto (280, 230);
 _lineto (280, 320);
 _moveto (221, 230);
 _lineto (221, 320);
 _moveto (160, 230);
 _lineto (160, 320);
 _moveto (356, 230);
 _lineto (356, 320);
 _moveto (10, 260);
 _lineto (430,260);

 display_result("Productivity (mg/h)",1,14);

 sprintf(buffer,"%.2f",cons_prod_nom.value); /* SetPoint */
 display_result(buffer,21,14);

 sprintf(buffer,"%.2f",prod_mod.sp);         /* Model */
 display_result(buffer,29,14);

 sprintf(buffer,"%.2f",cons_prod_real.sp);   /* Realised */
 display_result(buffer,37,14);


 sprintf(buffer,"%.2f",production.sp);    /* Mesured */
 display_result(buffer,46,14);


 display_result("Flow          (l/h)", 1,15);
 sprintf(buffer,"%.3f",qe_real.sp);
 display_result(buffer,37,15);            /* Realised */

 sprintf(buffer,"%.3f",qe_real.value);
 display_result(buffer,21,15);            /* Setpoint */
```

```
}


void DrawCompartment(int x, int y)
{
_moveto(x,y);
_lineto(x+60,y);
_lineto(x+60,y+40);
_lineto(x,y+40);
_lineto(x,y);
}

void DownArrow (int x, int y)
{
_moveto(x,y);
_lineto(x-3,y-3);
_moveto(x,y);
_lineto(x+3,y-3);
}

void UpArrow (int x, int y)
{
_moveto(x,y);
_lineto(x-3,y+3);
_moveto(x,y);
_lineto(x+3,y+3);
}


void RightArrow (int x, int y)
{
_moveto(x,y);
_lineto(x-3,y-3);
_moveto(x,y);
_lineto(x-3,y+3);
}

void LeftArrow (int x, int y)
{
_moveto(x,y);
_lineto(x+3,y+3);
_moveto(x,y);
_lineto(x+3,y-3);
}


result_principal()
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("Main screen");

    use_display_window();
    _clearscreen(_GWINDOW);

_settextcolor(12);
_settextposition (5+TVA,37+THA);
_outtext ("Biomass");
```

```
_settextposition (15+TVA,42+THA);
_outtext ("Waste");

_settextcolor(2);
_settextposition (5+TVA,20+THA);
_outtext ("O2");
_settextposition (15+TVA,25+THA);
_outtext ("CO2");

_settextcolor(15);

_setcolor (2);
_moveto(200+HA,80+VA);
_lineto(60+HA,80+VA);
_lineto (60+HA, 164+VA);
DownArrow (60+HA,163+VA);

_setcolor (12);
_moveto(200+HA,100+VA);
_lineto(80+HA,100+VA);
_lineto (80+HA, 164+VA);
RightArrow (199+HA,100+VA);

_setcolor (12);
_moveto (110+HA,300+VA);
_lineto(70+HA,300+VA);
_lineto(70+HA,205+VA);
UpArrow(70+HA,205+VA);

_setcolor (2);
_moveto (130+HA,280+VA);
_lineto(130+HA,220+VA);
_lineto(220+HA,220+VA);
_lineto (220+HA,110+VA);
UpArrow(220+HA,111+VA);

_setcolor (12);
_moveto (150+HA,280+VA);
_lineto(150+HA,190+VA);
_lineto(340+HA,190+VA);
RightArrow(339+HA,190+VA);


_setcolor (2);
_moveto (290+HA, 300+VA);
_lineto (220+HA,300+VA);
_lineto (220+HA, 220+VA);
UpArrow(220+HA,221+VA);

_setcolor (12);
_moveto(320+HA, 320+VA);
_lineto(320+HA, 340+VA);
_lineto(140+HA,340+VA);
_lineto(140+HA,320+VA);
UpArrow(140+HA,321+VA);

_setcolor (12);
_moveto(370+HA,204+VA);
_lineto(370+HA, 240+VA);
_lineto(320+HA, 240+VA);
_lineto(320+HA,280+VA);
DownArrow(320+HA,279+VA);

_setcolor (2);
```

```
_moveto (260+HA,100+VA);
_lineto (360+HA,100+VA);
_lineto(360+HA, 164+VA);
LeftArrow (261+HA,100+VA);

_setcolor (2);
_moveto (240+HA,110+VA);
_lineto (240+HA,174+VA);
_lineto(340+HA, 174+VA);
RightArrow (340+HA,174+VA);

_setcolor (12);
_moveto (260+HA,80+VA);
_lineto (380+HA,80+VA);
_lineto(380+HA, 164+VA);
DownArrow (380+HA,163+VA);

_setcolor(15);
DrawCompartment (200+HA, 70+VA);
DrawCompartment (41+HA, 164+VA);
DrawCompartment (341+HA, 164+VA);
DrawCompartment (110+HA, 280+VA);
DrawCompartment (290+HA, 280+VA);

_settextcolor (15);
_settextposition (6+TVA,29+THA);
_outtext ("iv");
_settextposition (12+TVA,9+THA);
_outtext ("iii");
_settextposition (12+TVA,46+THA);
_outtext ("CREW");
_settextposition (19+TVA,18+THA);
_outtext ("ii");
_settextposition (19+TVA,41+THA);
_outtext ("i");

_settextcolor (12);
_settextposition (9+TVA,10+THA);
_outtext ("NO3-");
_settextposition (17+TVA,9+THA);
_outtext ("NH4+");
_settextposition (21+TVA,27+THA);
_outtext ("Acids");
_settextposition (13+TVA,33+THA);
_outtext ("Biomass");

_settextcolor(2);
_settextposition (10+TVA,45+THA);
_outtext ("CO2");
_settextposition (8+TVA,30+THA);
_outtext ("CO2");

_settextcolor (15);

    use_message_window();

    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);

}

result_nitrogen()
{
#ifndef ADERSA
```

```
         void display_result(char *,short,short);
 #else
         void display_result();
 #endif
         char buffer[150];

         set_title ("Nitrifying compartment");

         use_display_window();
         _clearscreen(_GWINDOW);
         use_message_window();


         display_result ("Value", 27, 2);
         display_result ("SetPoint", 38, 2);

         _moveto (30, 100);
         _lineto (375,100);
         _moveto (190, 80);
         _lineto (190, 220);
         _moveto (280, 80);
         _lineto (280, 220);

         display_result("DO  (%)",7,4);
         display_result("pH",7,5);
         sprintf(buffer,"%.1f",NitrDo.value);
         display_result(buffer,27,4);
         sprintf(buffer,"%.1f",NitrpH.value);
         display_result(buffer,27,5);
         sprintf(buffer,"%.1f",NitrDo.sp);
         display_result(buffer,39,4);
         sprintf(buffer,"%.1f",NitrpH.sp);
         display_result(buffer,39,5);

         display_result("Temp. (§C) ",7,6);
         sprintf(buffer,"%.1f", NitrTemp.value);
         display_result(buffer,27,6);
         sprintf(buffer,"%.1f", NitrTemp.sp);
         display_result(buffer,39,6);

         display_result("Pressure (bar) ",7,7);
         sprintf(buffer,"%.1f", NitrPressure.value);
         display_result(buffer,27,7);
         sprintf(buffer,"%.1f", NitrPressure.sp);
         display_result(buffer,39,7);

         display_result("NH4+   (mg/l) ",7,8);
         sprintf(buffer,"%.1f", NitrAmCon.value);
         display_result(buffer,27,8);
         sprintf(buffer,"%.1f", NitrAmCon.sp);
         display_result(buffer,39,8);

         display_result("NO3- (mg/l) ",7,9);
         sprintf(buffer,"%.1f", NitrNitrCon.value);
         display_result(buffer,27,9);
         sprintf(buffer,"%.1f", NitrNitrCon.sp);
         display_result(buffer,39,9);

}


result_reactor3()
{
#ifndef ADERSA
```

```
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("Reactor III");

    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();


    sprintf(buffer,"Dades presentar no seleccionades.");
    display_result(buffer,20,5);

    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}

result_reactor4()
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("Reactor IV");

    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();


    sprintf(buffer,"Dades presentar no seleccionades.");
    display_result(buffer,20,5);

    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}



int init_output_file(FILE *fp, char *name)
{
if (!diskfull)
use_message_window();
_clearscreen(_GWINDOW);
_outtext("Initializing output files ...");
if (!diskfull)
    {
    fp=fopen(name,"rt");
    if ( fp == NULL)
            {
            fp=fopen(name,"wt");
            if ( fp == NULL)
                    { display_error ("Impossible to open output file
!!");
                      wait_time(1);
                     return 1;
                    }
                    else
```

```
                        {
                        char buffer[400];
                        sprintf (buffer, "cxa_moy, nit_moy, production.sp,
react.temp, cons_prod_real.sp,  qe_real.sp, qe_real.value, Fr.sp,
Eb.sp\n");
                        fwrite(buffer, 1, strlen(buffer), fp);
                        }
                }
        _outtext(" done.\n");
     }
     else _outtext (" canceled\n");
    _outtext(" done.\n");
    wait_time(1);
    _clearscreen (_GWINDOW);
    fclose (fp);
    return 0;
}



int init_log_file(void)
{
FILE *fp;
char bf[200];
time_t ltime;
struct  tm  *dt;


use_message_window();
_clearscreen(_GWINDOW);
_outtext("Initializing log file ...");

if (!diskfull)
    {
    fp=fopen("log.txt","at");
    if ( fp == NULL)
            {
            display_error ("Impossible to open output file !!");
            wait_time(1);
            return 1;
            }
        time(&ltime);
        dt=localtime(&ltime);
        dt->tm_mon++;
        sprintf(bf," %02d/%02d/%02d   %02d:%02d:%02d Msg : ",dt-
>tm_mday,
              dt->tm_mon,dt->tm_year,dt->tm_hour,dt->tm_min,dt-
>tm_sec);
        strcat (bf, "REINITIALIZING\n");
        fwrite(bf, 1, strlen(bf), fp);
        wait_time(1);
        fclose (fp);
    fp=fopen("techlog.txt","at");
    if ( fp == NULL)
            {
            display_error ("Impossible to open technical output file
!!");
            wait_time(1);
            return 1;
            }
        fwrite(bf,1,strlen(bf), fp);
        _outtext(" done.\n");
    }
    else _outtext(" canceled.\n");
    wait_time(1);
    fclose (fp);
    _clearscreen (_GWINDOW);
    return 0;
}



result_status()
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("System status");


    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();

    if (ALPHAFILTERcxa!=1) sprintf(buffer,"Cxa filter : %.2f",
ALPHAFILTERcxa);
        else sprintf(buffer,"Cxa filter : OFF");
    display_result(buffer,2,2);
    if (ALPHAFILTERnitrate!=1) sprintf(buffer,"Nitrate filter : %.2f",
ALPHAFILTERnitrate);
        else sprintf(buffer,"Nitrate filter : OFF");
    display_result(buffer,2,3);
    if (ALPHAFILTEREb!=1) sprintf(buffer,"Eb filter : %.2f",
ALPHAFILTEREb);
        else sprintf(buffer,"Eb filter : OFF");
    display_result(buffer,2,4);
    if (ALPHAFILTERpH != 1) sprintf(buffer,"Ph filter : %.2f",
ALPHAFILTERpH);
        else sprintf(buffer,"Ph filter : OFF");
    display_result(buffer,2,5);

    sprintf(buffer,"Sampling sp : %i m.", SAMP_SPIRU);
    display_result(buffer,2,7);

    sprintf(buffer,"Sampling lq : %i m.", SAMP_LIQUE);
    display_result(buffer,2,8);

    sprintf(buffer,"Sampling rh : %i m.", SAMP_RHODO);
    display_result(buffer,2,9);

    sprintf(buffer,"Sampling nt : %i m.", SAMP_NITRI);
    display_result(buffer,2,10);

    sprintf (buffer, "Alarms : ");
    if (activated_alarms) strcat (buffer, "ON");
       else strcat (buffer, "OFF");
    display_result(buffer,35,2);

    if (diskwarning) {
            strcpy (buffer,"WARNING !! DISK TOO FULL");
            display_result(buffer,35,3);
            }
            else
            {
```

```
            strcpy (buffer,"Disk status : Ok.");
            display_result(buffer,35,3);
            }
    if (diskfull)
        {
        strcpy (buffer,"WARNING !! DISK FULL");
        display_result(buffer,35,4);
        strcpy (buffer,"WRITING DISABLED");
        display_result(buffer,39,5);
        }

    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}

void move_disk (void)
{
char buffer[150];
char name[40], destination[40];
unsigned long lliure;

set_title ("Floppy backups");

use_display_window();
_clearscreen(_GWINDOW);
use_message_window();

sprintf(buffer,"Please, insert a floppy disk in drive A: and ");
center_display_result(buffer,3);
sprintf(buffer,"type the name of the file you want to backup.");
center_display_result(buffer,4);

sprintf(buffer,"The syntax to use is : xxyymmdd where ...");
center_display_result(buffer,6);
sprintf(buffer,"xx is the reactor name sp, nt, lq or rh.");
center_display_result(buffer,7);
sprintf(buffer,"yy is the year.");
center_display_result(buffer,8);
sprintf(buffer,"yy is the month.");
center_display_result(buffer,9);
sprintf(buffer,"yy is the day.");
center_display_result(buffer,10);

sprintf(buffer,"F1 - Help");
center_display_result(buffer,20);

sprintf(buffer,"File name : ");
display_result(buffer,15,13);

strcpy (buffer, "");
if (!get_input_string (name, 27 ,13, 39)) return;
if (strlen (name) > 8 || strlen (name) == 0
    || (!(toupper (name[0]) == 'S' && toupper (name[1]) == 'P') &&
        !(toupper (name[0]) == 'N' && toupper (name[1]) == 'T')) )
                    {
                    display_status ("This is not a valid file name");
                    printf("\a\a");
                    wait_time(2);
                    display_status (" ");
                    return;
                    }

sprintf (buffer, "%s.out",name);      /* Complete the file name */
sprintf (destination, "a:%s",buffer);
```

```
strcpy (name, buffer);

move_files (name, destination);


display_status (" ");
}


void copy_disk (void)
{
char buffer[150];
char name[40], destination[40];
unsigned long lliure;

set_title ("Floppy backups");

use_display_window();
_clearscreen(_GWINDOW);
use_message_window();

sprintf(buffer,"Please, insert a floppy disk in drive A: and ");
center_display_result(buffer,3);
sprintf(buffer,"type the name of the file you want to backup.");
center_display_result(buffer,4);

sprintf(buffer,"The syntax to use is : xxyymmdd where ...");
center_display_result(buffer,6);
sprintf(buffer,"xx is the reactor name sp, nt, lq or rh.");
center_display_result(buffer,7);
sprintf(buffer,"yy is the year.");
center_display_result(buffer,8);
sprintf(buffer,"yy is the month.");
center_display_result(buffer,9);
sprintf(buffer,"yy is the day.");
center_display_result(buffer,10);

sprintf(buffer,"F1 - Help");
center_display_result(buffer,20);

sprintf(buffer,"File name : ");
display_result(buffer,15,13);

strcpy (buffer, "");
if (!get_input_string (name, 27 ,13, 40)) return;
if (strlen (name) > 8 || strlen (name) == 0
    || ( !(toupper (name[0]) == 'S' && toupper (name[1]) == 'P') &&
         !(toupper (name[0]) == 'N' && toupper (name[1]) == 'T') ))
                        {
                        display_status ("This is not a valid file name");
                        printf("\a\a");
                        wait_time(2);
                        display_status (" ");
                        return;
                        }

sprintf (buffer, "%s.out",name);      /* Complete the file name */
sprintf (destination, "a:%s",buffer);
strcpy (name, buffer);


lliure = checkfloppy();
if (lliure == -1) {
                display_status ("Drive A: is not ready.");
                printf ("\a\a");
                wait_time(2);
                display_status (" ");
                return;
                }

if (copia(name, destination, lliure)!=0)
        {
        display_status ("An error occurred while copying ...");
        printf ("\a\a");
        }

display_status (" ");
}


void file_menu(void)
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];

    set_title ("File options");

    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();

    sprintf(buffer,"F5 - Move to floppy output files.");
    display_result(buffer,20,5);
    sprintf(buffer,"F6 - Copy to floppy log file.");
    display_result(buffer,20,6);
    sprintf(buffer,"F7 - Remove log file.");
    display_result(buffer,20,7);
    sprintf(buffer,"F8 - Directory.");
    display_result(buffer,20,8);
    sprintf(buffer,"F9 - Copy to floppy output files.");
    display_result(buffer,20,9);

    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}


void directory(void)
{
#ifndef ADERSA
        void display_result(char *,short,short);
#else
        void display_result();
#endif
        char buffer[150];
        struct find_t fi;
        int x=3, y=3;
        struct diskfree_t df;
        unsigned long lliure;

    set_title ("Directory");

    use_display_window();
    _clearscreen(_GWINDOW);
    use_message_window();


    if (_dos_findfirst("*.out",_A_NORMAL,&fi) == 0)
        {
        sprintf (buffer, "%- 12s % 6li bytes", fi.name, fi.size);
        display_result(buffer, x, y);
        while (_dos_findnext(&fi)==0)
            {
            if ((++y) == 20) {
                                display_status ("Press a key ...");
                                while (!kbhit());
                                getch();
                                use_display_window();
                                _clearscreen(_GWINDOW);
                                use_message_window();
                                sprintf(buffer,"F1 - Help");
                                center_display_result(buffer,20);
                                x = 3; y = 3;
                                }
            sprintf (buffer, "%- 12s % 6li bytes \n", fi.name,
fi.size);
            display_result(buffer, x, y);
            }
        }

        if ((++y) == 20) {
                            display_status ("Press a key ...");
                            while (!kbhit());
                            getch();
                            use_display_window();
                            _clearscreen(_GWINDOW);
                            use_message_window();
                            sprintf(buffer,"F1 - Help");
                            center_display_result(buffer,20);
                            x = 3; y = 3;
                            }


    if (_dos_findfirst("*.txt",_A_NORMAL,&fi) == 0)
        {
        sprintf (buffer, "%- 12s % 6li bytes", fi.name, fi.size);
        display_result(buffer, x, y);
        while (_dos_findnext(&fi)==0)
            {
            if ((++y) == 20) {
                                    display_status ("Press a key ...");
                                    while (!kbhit());
                                    getch();
                                    use_display_window();
                                    _clearscreen(_GWINDOW);
                                    use_message_window();
                                    sprintf(buffer,"F1 - Help");
                                    center_display_result(buffer,20);
                                    x = 3; y = 3;
                                }
            sprintf (buffer, "%- 12s % 6li bytes \n", fi.name,
fi.size);
            display_result(buffer, x, y);
            }
        }

    _dos_getdiskfree(0, &df);    /* Ud per defecte */
    lliure = (unsigned long) df.avail_clusters *
```

```
                (unsigned long)  df.sectors_per_cluster *
                (unsigned long) df.bytes_per_sector;


    sprintf (buffer,"*.out, *.txt %li bytes free", lliure);
    display_status (buffer);
    sprintf(buffer,"F1 - Help");
    center_display_result(buffer,20);
}
```

Screen.c

```
/*********************************************************************

NAME            SCREEN.C

AUTHOR          BINOIS C & PEDRO L. PONS

DESCRIPTION
      screen and graphic functions listing file

UPDATES
    06-05-93, 11-05-97

*********************************************************************/


#include <stdio.h>
#include <graph.h>
#include <string.h>
#include <stdlib.h>

#include "melissa.h"
#include "results.h"
#include "screen.h"

void set_title(char*);
extern int next_pfc_rh;
extern int next_pfc_lq;


static double x=41;
static int Xo=80, Yo=300;
static int Heigth = 200;     /* Heigth of the graphic */
static int Length = 320;     /* Length of the graphic */
static datagraph spdata, ntdata, simdata1, simdata2, simdata3;
static datagraph *actdata;
static char olddate[12], olddateend[12];
static int lastscale = 0;
long CalculateNextDate(long);


/*-------------------------------------
      screen initialisation
-------------------------------------*/

void screen_init(void)
      {
    _setvideomode(_VRES16COLOR);
    _setbkcolor((long)RED);
    _settextwindow(1,1,25,80);
    _clearscreen(_GWINDOW);

    _settextcolor(BLACK);
    _settextcolor(5);

      _setbkcolor((long)BLUE);
      _settextcolor(WHITE);
#ifdef __Monitoring
    _settextposition(2, 59);
    _settextcolor(4);
    _outtext(" Monitoring station ");
#else
```

```
     _settextposition(2, 60);
     _settextcolor(4);
     _outtext(" Control station ");
 #endif
     _settextposition (2,2);
     _settextcolor(11);
     _outtext(" MELISSA GPS V2.2e  ");

     _setcolor(1);
     _moveto(450, 0);
     _lineto (450,392);
     _moveto (640, 392);
     _lineto (0,392);
     _moveto(180, 39);
     _lineto (180,0);
     _moveto (0,0);
     _lineto (639,0);
     _lineto (639,473);
     _lineto (0, 473);
     _lineto (0,0);
     _moveto (639,39);
     _lineto (0,39);
     _setcolor (7);

     _settextwindow(4,2,24,55);

     _clearscreen(_GWINDOW);

     _settextwindow(26,2,29,78);
     _clearscreen(_GWINDOW);
     _settextwindow (4,58,24,79);
     _clearscreen(_GWINDOW);

      use_message_window();
      _wrapon(_GWRAPON);
      _displaycursor(_GCURSOROFF);

      }

/*-------------------------------------
      display activ group
-------------------------------------*/

display_activ_group(GPS_FILE *gps)
      {
      char    buffer[100];
      struct  rccoord txtpos;
      txtpos=_gettextposition();

      use_group_window();
    sprintf(buffer,"GROUP : %s",gps->file);
      _outtext(buffer);

      use_message_window();
      _settextposition(txtpos.row,txtpos.col);
      }

/*-------------------------------------
      display no activ group
-------------------------------------*/

display_no_activ_group()
      {
      char    buffer[100];
```

```
       struct  rccoord txtpos;
       txtpos=_gettextposition();

       use_group_window();
     sprintf(buffer,"GROUP  : --------.GPS");
       _outtext(buffer);

       use_message_window();
       _settextposition(txtpos.row,txtpos.col);
       }


/*--------------------------------------
      display current time
--------------------------------------*/

display_time(char *buffer)
      {
      struct  rccoord txtpos;
      txtpos=_gettextposition();

      use_time_window();
      _outtext(buffer);

      use_message_window();
      _settextposition(txtpos.row,txtpos.col);
      }

/*--------------------------------------
      display result in main window
--------------------------------------*/

display_result(char *buffer,short x,short y)
      {
      struct  rccoord txtpos;

    if((x>55)||(y>20))
            {
            display_error("Can't display result : coordinates error on
:");
            display_error(buffer);
            return(-1);
            }
      txtpos=_gettextposition();

      use_display_window();
      _settextposition(y,x);
      _outtext(buffer);

      use_message_window();
      _settextposition(txtpos.row,txtpos.col);
      }

center_display_result(char *buffer, short y)
      {
      struct  rccoord txtpos;
    int l;

    l = strlen(buffer);

    if(y>20 || l > 52)
            {
      display_error("Can't center display result : coordinates error
on :");
```

```
            display_error(buffer);
            return(-1);
            }
        txtpos=_gettextposition();

     l = (int) l / 2;

        use_display_window();
      _settextposition(y,26-l);
        _outtext(buffer);

        use_message_window();
        _settextposition(txtpos.row,txtpos.col);
     }

/* ============================
     Routine for getting an input
     string or a number from the keyboard.
    ==============================*/

int InputKey (char* buf, int maxim, int onlynumbers)
{
char c;
int ndx=0;

c = getch();
while (c!=13 && c!=27 && ndx < maxim)
     {
     if(onlynumbers!=0)
       {
       if ( (c >= '0' && c<='9')|| c==8 )
       {
       buf[ndx++]=c;
       printf ("%c",c);
       }
       }
       else
       {
       buf[ndx++]=c;
       printf ("%c",c);
       }
     if (c==8) {
             printf (" \b");
             ndx-=2;
             buf[ndx]='\0';
          }
     c=getch();
     }
buf[ndx]='\0';
if (c==27) buf[0]=0;
if (!ndx || c==27) return 0;
return 1;
}


/* ===================================
Gets a string from the data window
=================================== */

int get_input_string (char *buffer,short x,short y, int maxim)
      {
      struct  rccoord txtpos;
    int rt;
    if((x>52)||(y>20))
            {
            display_error("Can't display result : coordinates error on
:");
            display_error(buffer);
            return(-1);
            }
      txtpos=_gettextposition();

      use_display_window();
      _settextposition(y,x);

    rt=InputKey(buffer, maxim ,0);

      use_message_window();
      _settextposition(txtpos.row,txtpos.col);
    return rt;
    }


/* ==================================
    Gets a number from the data window
================================== */

int get_input_number (char *buffer , short x,short y, int max)
      {
    int rt;
    struct  rccoord txtpos;

    if((x>52)||(y>20))
            {
            display_error("Can't display result : coordinates error on
:");
            display_error(buffer);
            return(-1);
            }
      txtpos=_gettextposition();

      use_display_window();
    _settextposition(y,x);

    rt=InputKey(buffer, max ,1);

      use_message_window();
      _settextposition(txtpos.row,txtpos.col);
    return rt;
    }


/*--------------------------------------
      display system status
--------------------------------------*/

display_status(char *buffer)
      {
      struct  rccoord txtpos;
      txtpos=_gettextposition();

      use_status_window();
      _outtext(buffer);

      use_message_window();
      _settextposition(txtpos.row,txtpos.col);
```

```
        }


/*-------------------------------------
       display error messages
-------------------------------------*/

display_error(char *buffer)
       {
       FILE *fp;


       _setbkcolor((long)GREEN);
       _settextcolor(RED+16);
       _outtext(buffer);
       printf("\a\a\a");
       _setbkcolor((long)BLUE);
       _settextcolor(WHITE);
       _wrapon(_GWRAPON);
       _displaycursor(_GCURSOROFF);
     error_log_file(buffer);
     _outtext("\n");

     }

/*-------------------------------------
       use messages area
-------------------------------------*/

use_message_window()
       {
     _settextwindow(26,2,29,78);
     _setbkcolor((long)BLUE);
       _settextcolor(WHITE);
       _wrapon(_GWRAPON);
       _displaycursor(_GCURSOROFF);
       }

/*-------------------------------------
       use group display area
-------------------------------------*/

use_group_window()
       {
     _settextwindow(5,59,5,79);
       tab(3,4);
       _settextcolor(WHITE);
       _setbkcolor((long)BLUE);
       _clearscreen(_GWINDOW);
       }

/*-------------------------------------
       use time display area
-------------------------------------*/

use_time_window()
       {
     _settextwindow(4,59,4,79);
       tab(58,4);
       _settextcolor(WHITE);
       _setbkcolor((long)BLUE);
       }

/*-------------------------------------
```

```
       use status display area
-------------------------------------*/

use_status_window()
       {
     _settextwindow(24,2,24,55);
     tab(3,18);
       _settextcolor(WHITE);
       _setbkcolor((long)BLUE);
       _clearscreen(_GWINDOW);
       }

/*-------------------------------------
       use main display area
-------------------------------------*/

use_display_window()
       {
     _settextwindow(5,2,23,55);
     _settextcolor(WHITE);
       _setbkcolor((long)BLUE);
       tab(3,6);
       }

/*-------------------------------------
       use main display area
-------------------------------------*/

void set_title(char *tl)
       {
     int i;
     char bf[100];
     i = strlen(tl);
     if (i > 29) return;
     i = (int) i / 2;
     _settextwindow(2,25,2,55);
     _clearscreen(_GWINDOW);
     _settextposition(0, 15-i);
     _outtext (tl);
     use_message_window();
     }

/*-------------------------------------
       move the cursor to the position (x,y)
-------------------------------------*/

tab(short x,short y)
       {
       _settextposition(y,x);
       }

/* -----------------------------------
     Use the control display area
---------------------------------- */
use_control_window()
       {
     _settextwindow(6,58,24,79);
     tab(3,18);
       _settextcolor(WHITE);
       _setbkcolor((long)BLUE);
       _clearscreen(_GWINDOW);
     }

/* ---------------------------------
```

```
    Display the next control minute.
---------------------------------- */

void next_control (void)
{
struct  rccoord txtpos;
char buffer[100];

txtpos=_gettextposition();

use_control_window();
sprintf (buffer, "Control sp : %i min.", next_pfc_sp);
_settextposition(2,2);
_outtext(buffer);
sprintf (buffer, "Control nt : %i min.", next_pfc_nt);
_settextposition(3,2);
_outtext(buffer);
sprintf (buffer, "Control lq : %i min.", next_pfc_lq);
_settextposition(4,2);
_outtext(buffer);
sprintf (buffer, "Control rh : %i min.", next_pfc_rh);
_settextposition(5,2);
_outtext(buffer);

use_display_window();
_settextposition(txtpos.row,txtpos.col);

}

/*========================================

    Here begins the graphic representation
    of data.
====================================== */

/*-----------------------------------
    This function reads the configuration
    file for the spiruline compartment graphic
    and sets the parameters.
-----------------------------------*/


void InitializeSpPlot(void)
{
FILE *fp;
int i;
int v;

fp = fopen ("spgraph.cfg","rt");
if (fp==NULL)
    {
    spdata.activ[0] = 0;
    spdata.activ[1] = 0;
    spdata.activ[2] = 0;
    spdata.activ[3] = 0;
    spdata.activ[4] = 0;
    spdata.activ[5] = 0;
    spdata.activ[6] = 0;
    spdata.activ[7] = 0;
    spdata.activ[8] = 0;
    return;
    }
for (i=0; i < NUMVARSTOPLOT && ! feof (fp); i++)
    {
```

```
    v=fscanf (fp, "%i %i %f %f %49s", &spdata.activ[i],
&spdata.colors[i],  &spdata.min[i], &spdata.max[i], spdata.name[i]);
    if (v != 5)
      {
      char buf[100];
      sprintf(buf, "Error in configuration file, line %i", i);
      display_error (buf);
      spdata.activ[0] = 0;
      spdata.activ[1] = 0;
      spdata.activ[2] = 0;
      spdata.activ[3] = 0;
      spdata.activ[4] = 0;
      spdata.activ[5] = 0;
      spdata.activ[6] = 0;
      spdata.activ[7] = 0;
      spdata.activ[8] = 0;
      i=NUMVARSTOPLOT;
      }
    }
fclose(fp);
}


void InitializeNtPlot(void)
{
FILE *fp;
int i;
int v;

fp = fopen ("ntgraph.cfg","rt");
if (fp==NULL)
    {
    ntdata.activ[0] = 0;
    ntdata.activ[1] = 0;
    ntdata.activ[2] = 0;
    ntdata.activ[3] = 0;
    ntdata.activ[4] = 0;
    ntdata.activ[5] = 0;
    ntdata.activ[6] = 0;
    ntdata.activ[7] = 0;
    ntdata.activ[8] = 0;
    return;
    }
for (i=0; i < NUMVARSTOPLOT && ! feof (fp); i++)
    {
    v=fscanf (fp, "%i %i %f %f %49s", &ntdata.activ[i],
&ntdata.colors[i],  &ntdata.min[i], &ntdata.max[i], ntdata.name[i]);
    if (v != 5)
      {
      char buf[100];
      sprintf(buf, "Error in configuration file, line %i", i);
      display_error (buf);
      ntdata.activ[0] = 0;
      ntdata.activ[1] = 0;
      ntdata.activ[2] = 0;
      ntdata.activ[3] = 0;
      ntdata.activ[4] = 0;
      ntdata.activ[5] = 0;
      ntdata.activ[6] = 0;
      ntdata.activ[7] = 0;
      ntdata.activ[8] = 0;
      i=NUMVARSTOPLOT;
      }
    }
```

```
fclose(fp);
}


void InitializeSim1Plot(void)
{
FILE *fp;
int i;
int v;

fp = fopen ("sm1graph.cfg","rt");
if (fp==NULL)
    {
    simdata1.activ[0] = 0;
    simdata1.activ[1] = 0;
    simdata1.activ[2] = 0;
    simdata1.activ[3] = 0;
    simdata1.activ[4] = 0;
    simdata1.activ[5] = 0;
    simdata1.activ[6] = 0;
    simdata1.activ[7] = 0;
    simdata1.activ[8] = 0;
    return;
    }
for (i=0; i < NUMVARSTOPLOT && ! feof (fp); i++)
    {
    v=fscanf (fp, "%i %i %f %f %49s", &simdata1.activ[i],
&simdata1.colors[i],  &simdata1.min[i], &simdata1.max[i],
simdata1.name[i]);
    if (v != 5)
      {
      char buf[100];
      sprintf(buf, "Error in configuration file, line %i", i);
      display_error (buf);
      simdata1.activ[0] = 0;
      simdata1.activ[1] = 0;
      simdata1.activ[2] = 0;
      simdata1.activ[3] = 0;
      simdata1.activ[4] = 0;
      simdata1.activ[5] = 0;
      simdata1.activ[6] = 0;
      simdata1.activ[7] = 0;
      simdata1.activ[8] = 0;
      i=NUMVARSTOPLOT;
      }
    }

fclose(fp);
}

void InitializeSim2Plot(void)
{
FILE *fp;
int i;
int v;

fp = fopen ("sm2graph.cfg","rt");
if (fp==NULL)
    {
    simdata2.activ[0] = 0;
    simdata2.activ[1] = 0;
    simdata2.activ[2] = 0;
    simdata2.activ[3] = 0;
    simdata2.activ[4] = 0;
    simdata2.activ[5] = 0;
    simdata2.activ[6] = 0;
    simdata2.activ[7] = 0;
    simdata2.activ[8] = 0;
    return;
    }
for (i=0; i < NUMVARSTOPLOT && ! feof (fp); i++)
    {
    v=fscanf (fp, "%i %i %f %f %49s", &simdata2.activ[i],
&simdata2.colors[i], &simdata2.min[i], &simdata2.max[i],
simdata2.name[i]);
    if (v != 5)
      {
      char buf[100];
      sprintf(buf, "Error in configuration file, line %i", i);
      display_error (buf);
      simdata2.activ[0] = 0;
      simdata2.activ[1] = 0;
      simdata2.activ[2] = 0;
      simdata2.activ[3] = 0;
      simdata2.activ[4] = 0;
      simdata2.activ[5] = 0;
      simdata2.activ[6] = 0;
      simdata2.activ[7] = 0;
      simdata2.activ[8] = 0;
      i=NUMVARSTOPLOT;
      }
    }

fclose(fp);
}

void InitializeSim3Plot(void)
{
FILE *fp;
int i;
int v;

fp = fopen ("sm3graph.cfg","rt");
if (fp==NULL)
    {
    simdata3.activ[0] = 0;
    simdata3.activ[1] = 0;
    simdata3.activ[2] = 0;
    simdata3.activ[3] = 0;
    simdata3.activ[4] = 0;
    simdata3.activ[5] = 0;
    simdata3.activ[6] = 0;
    simdata3.activ[7] = 0;
    simdata3.activ[8] = 0;
    return;
    }
for (i=0; i < NUMVARSTOPLOT && ! feof (fp); i++)
    {
    v=fscanf (fp, "%i %i %f %f %49s", &simdata3.activ[i],
&simdata3.colors[i], &simdata3.min[i], &simdata3.max[i],
simdata3.name[i]);
    if (v != 5)
      {
      char buf[100];
      sprintf(buf, "Error in configuration file, line %i", i);
      display_error (buf);
      simdata3.activ[0] = 0;
      simdata3.activ[1] = 0;
```

```
        simdata3.activ[2]  = 0;
        simdata3.activ[3]  = 0;
        simdata3.activ[4]  = 0;
        simdata3.activ[5]  = 0;
        simdata3.activ[6]  = 0;
        simdata3.activ[7]  = 0;
        simdata3.activ[8]  = 0;
        i=NUMVARSTOPLOT;
        }
    }

fclose(fp);
}


/* --------------------------------
    This function prints the legend.
 -------------------------------- */
void DrawLegend (void)
{
int used=0,
    base=17,
    xbase = 0, i;


for (i=0; i < NUMVARSTOPLOT; i++)
    {
    if (actdata->activ[i])
      {
      use_display_window();
      _settextposition (base+used, 4+xbase);
      _settextcolor(actdata->colors[i]);
      _outtext(actdata->name[i]);
      _settextcolor(15);
      use_message_window();
      used++;
      if (used==3) { used = 0; base = 17; xbase +=17; }
      }
    }

use_message_window();

}

/* ----------------------------------
    This function draws the scale
    for the "i" variable
------------------------------------- */

void DrawScale(int i)
{
char buffer[100];

use_display_window();
_settextcolor(actdata->colors[i]);
_settextposition(3,0);
_outtext("       ");
_settextposition(3,0);
sprintf (buffer, "%4.2f", actdata->max[i]);
_outtext(buffer);
_settextposition(6,0);
_outtext("       ");
_settextposition(6,0);
sprintf (buffer, "%4.2f", 3*(actdata->max[i]/4));
```

```
_outtext(buffer);
_settextposition(9,0);
_outtext("      ");
_settextposition(9,0);
sprintf (buffer, "%4.2f", actdata->max[i]/2);
_outtext(buffer);
_settextposition(12,0);
_outtext("      ");
_settextposition(12,0);
sprintf (buffer, "%4.2f", actdata->max[i]/4);
_outtext(buffer);
_settextcolor(15);

_setcolor(15);
_moveto(Xo-3,Yo-Heigth);
_lineto(Xo+2,Yo-Heigth);

use_message_window();
}

void AdvanceScale (void)
{
int i;

i = lastscale;

do
    {
    i = (i+1) % NUMVARSTOPLOT;
    }
    while (!actdata->activ[i] && i != lastscale );
lastscale = i;
DrawScale(lastscale);
}

void GoBackScale (void)
{
int i;

i = lastscale;

do
    {
    i = --i;
    if (i<0) i = NUMVARSTOPLOT-1;
    i = i % NUMVARSTOPLOT;
    }
    while (!actdata->activ[i] && i != lastscale );
lastscale = i;
DrawScale(lastscale);
}


/* ----------------------------------
    This one draws the axes ...
---------------------------------- */

void DrawAxes (int np, long beg, long end)
{
char cad[100];

_setcolor (15);
_moveto (Xo, Yo);
_lineto (Xo+(Length*1.1), Yo);
```

```
_moveto (Xo, Yo);
_lineto (Xo, Yo - (Heigth *1.1));
_moveto (Xo-1, Yo+1);
_lineto (Xo+(Length*1.1), Yo+1);
_moveto (Xo-1, Yo);
_lineto (Xo-1, Yo - (Heigth *1.1));
_moveto(Xo-3,Yo - Heigth + (Heigth/2));
_lineto (Xo+2, Yo - Heigth + (Heigth/2) );
_moveto(Xo-3,Yo - Heigth + 3*(Heigth/4));
_lineto (Xo+2, Yo - Heigth + 3*(Heigth/4) );
_moveto(Xo-3,Yo - Heigth + (Heigth/4));
_lineto (Xo+2, Yo - Heigth + (Heigth/4) );
use_display_window();
_settextposition (16,4);
_settextcolor(11);
_outtext(olddate);
_settextposition (16,47);
_outtext (olddateend);
use_message_window();
}


/* ----------------------------------
    This function, given a file name
    and a number of points, reads the
    data from de file and prints the
    graphic of that file.
---------------------------------- */

int ReadDraw (char *name, int np)
{
FILE *fp;
char c[2], input[300];
char hora[6], cr;
int line=0;
int first=6;     /* N§ of variables to graph */
double delta;     /* pixel per point */
int i;

double loc;

delta = Length / (double) np;

c[1]='\0';
strcpy (input, "");

_setcolor(15);
_moveto (x, Yo-2);
_lineto (x, Yo+3);

fp = fopen (name, "rt");
if (fp==NULL) {
            char buffer[100];
            sprintf(buffer, "Error opening file : %s", name);
            display_status(buffer);
            return -1;
            }
    else {
        read (&cr,1,1,fp);
        while (cr != '\n') fread (&cr, 1,1,fp);
        while (!feof(fp))
          {
          fread (&c[0], 1, 1, fp);
          if (ferror (fp)) {
            char buffer[100];
            sprintf(buffer, "Error opening file : %s", name);
            display_status(buffer);
            return -1;
            }
          if (c[0] == '\n')    /* let's read all a line ... */
            {
            line++;


            sscanf (input, "%5c %f %f %f %f %f %f %f %f %f",hora,
&actdata->v[0], &actdata->v[1], &actdata->v[2], &actdata->v[3],
&actdata->v[4], &actdata->v[5], &actdata->v[6], &actdata->v[7],
&actdata->v[8]);
            for (i=0; i < 9; i++)
              {
                if (actdata->activ[i] ) {
                if (!first) {
                        if ( Yo-Heigth < actdata->y[i] )
                             _moveto (x,actdata->y[i]);
                             else
                             _moveto ((int)x+delta,Yo-Heigth);
                        }
                  else
                     {
                     if (Yo - (((actdata->v[i]-actdata-
>min[i])/actdata->max[i]) * Heigth)  > Yo - Heigth)
                         {_moveto(x, Yo - (((actdata->v[i]-actdata-
>min[i])/actdata->max[i]) * Heigth) );}
                         else
                         {_moveto(x, Yo-Heigth); }
                     first--;
                     }
                _setcolor (actdata->colors[i]);
                actdata->y[i] = actdata->v[i];
                actdata->y[i] = ((actdata->y[i]-actdata->min[i]) /
actdata->max[i]) * Heigth;
                actdata->y[i] = Yo - actdata->y[i];
                if ( Yo-Heigth < actdata->y[i] )
                  _lineto ((int)x+delta,actdata->y[i]);
                  else
                  _lineto ((int)x+delta,Yo-Heigth);
                }
              }
            x += delta;
            strcpy (input, "");      /* Reinitialize input string */
            }
            else
            {
            strcat (input, c);
            }

          }

        fclose(fp);
        return line-2;     /* Return the n§ of points of the file
(last, first) */
      }
}


/* --------------------------------
    This calculates the names of
    the files and and order to
    draw them
---------------------------------- */

int Draw (int np, long beg, long end)
{
long k;
int r;
char cad[40];

x = Xo;

if (actdata->filename[0] != 's' || actdata->filename[1] != 'i' ||
    actdata->filename[2] != 'm') {
            /* if it is not a simulation */
      for (k=beg; k <= end; k = CalculateNextDate (k))
          {
          sprintf (cad, "/gpsdat/%s%li.out", actdata->filename, k);
          r = ReadDraw(cad,np);
          if (r==-1)
                {
                return -1;
                }
          }
      } else {         /* if it is a simulation */
      if (actdata->filename[3] == '1') strcpy (cad,
Simulation1FileName);
      if (actdata->filename[3] == '2') strcpy (cad,
Simulation2FileName);
      if (actdata->filename[3] == '3') strcpy (cad,
Simulation3FileName);
      r = ReadDraw(cad,np);
      if (r==-1) {
            return -1;
            }
      }
return 0;

}


/* ------------------------------
    The next routine, given a file
    name, test its existence and check
    the number of lines of the file
-----------------------------------*/

int testfileexist (char *name)
{
FILE *fp;
char c[2],
     hora[6],
     input[300];

int i;
int line=0;

c[1] = '\0';

strcpy (input, "");
fp = fopen (name, "rt");
if (fp==NULL) return -1;
    else {
        while (!feof(fp))
          {
```

```
          fread (&c[0], 1, 1, fp);
          if (c[0] == '\n')
            {
            line++;
            if (line >1)
                {
                /* Check for maximum */
                sscanf (input, "%5c %f %f %f %f %f %f %f %f %f",hora,
                  &actdata->v[0], &actdata->v[1], &actdata->v[2],
&actdata->v[3],
                  &actdata->v[4], &actdata->v[5], &actdata->v[6],
&actdata->v[7],
                  &actdata->v[8]);
/*                    for (i=0; i < NUMVARSTOPLOT; i++)
                  {
                  if (actdata->v[i] > actdata->max[i]) actdata->max[i]
= actdata->v[i];
                  }*/
                strcpy (input,"");
                }
            }
            else {
            strcat(input, c);
            }
          }

        fclose(fp);
        return line-2;     /* Return the n§ of points of the file
(last, first) */
      }
}

/* ============================
    This routine calculate the
    next date given one.
   ============================ */

long CalculateNextDate(long a)
{
long  month ,year, day;

day = a %10;
a -= (a % 10);
a /= 10;
day += (a % 10)*10;

a -= (a % 10);
a /= 10;

month = a % 10;
a -= (a%10);
a /= 10;

month += (a % 10)*10;    /* Calculate the month */
a -= (a % 10);
a /= 10;

year = (a%10);
a -= (a %10);
a /= 10;
year += (a % 10)*10;

if (day==28 && month==2) return year*10000+(month+1)*100+1;
if ((day == 31) && ( month == 1 || month == 3 ||
                month == 5 || month == 7 ||
                month == 8 || month == 10 ) )
                 return year*10000 + (month+1)*100 + 1;
if ((day == 30) && ( month == 4 || month == 6 ||
                month == 6 || month == 9 ||
                month == 11 )) return year*10000 + (month+1)*100 + 1;

if (day==31 && month == 12) return (year+1)*10000+101;

return year*10000+month*100+day+1;
}



/* -------------------------
    This routine calculate the
    number of files between two dates
    and test them.
----------------------------*/

int calcule_between (long beg, long end)
{
long k;
long lines=0, lf;               /* lf : number of lines per file */
char cad[100], cad2[100];

/*if (abs (end - beg) > 100) return -1;     /* Only graphics for 15
days */

_outtext ("\n\n");

for (k=beg; k <= end; k = CalculateNextDate (k))
    {
    sprintf (cad, "/gpsdat/%s%li.out",actdata->filename, k);
            /* Test if file exist */
    lf = (long) testfileexist(cad);
    if (lf==-1) {
            return -1;
            }
      else
      lines = lines + lf;
    }

return lines;
}

/* ------------------------------
    This is the routine to be called
    when a graphical presentation is
    wished.
  --------------------------------- */
int drawgraph (int new, int interv)
{
char cad[100], cad2[100];
long begin, end, nlines;
time_t ltime;
struct  tm  *dt;

use_display_window();
_clearscreen(_GWINDOW);
use_message_window();

if (actdata->filename[0] != 's' || actdata->filename[1] != 'i' ||
    actdata->filename[2] != 'm') {
      if (new) {       /* if it is not the simulation */
          actdata->initialisation();
          display_result ("Initial date : (yymmdd) ",5,8);
          use_display_window();
          _settextposition(8,30);
          if (!get_input_number(cad, 30, 8, 100)) return 0;
          if (interv) {
            display_result ("Ending date : (yymmdd) ",5,9);
            use_display_window();
            _settextposition(9,30);
            if (!get_input_number(cad2, 30,9, 100)) return 0;
            } else {
            time(&ltime);
            dt=localtime(&ltime);
            dt->tm_mon++;
            sprintf(cad2,"%02d%02d%02d",dt->tm_year,dt->tm_mon,dt-
>tm_mday);
            }
           use_message_window();
           } else {
           strcpy (cad, olddate);
           if (!interv) {
                time(&ltime);
                dt=localtime(&ltime);
                dt->tm_mon++;
                sprintf(cad2,"%02d%02d%02d",dt->tm_year,dt->tm_mon,dt-
>tm_mday);
                } else {
                strcpy (cad2, olddateend);
                }
          }
      strcpy (olddate, cad);
      strcpy (olddateend, cad2);
      begin = atol(olddate);
      end = atol (olddateend);
      nlines = calcule_between (begin, end);  /* Get the number of
points to draw */
      } else {        /* if is the simulation ... */
      if (new) {
            actdata->initialisation();
            }
      if (actdata->filename[3] == '1') strcpy (cad,
Simulation1FileName);
      if (actdata->filename[3] == '2') strcpy (cad,
Simulation2FileName);
      if (actdata->filename[3] == '3') strcpy (cad,
Simulation3FileName);
      nlines = (long) testfileexist(cad);             /* Count the
numer of lines to plot */
      strcpy (olddate, "----");
      strcpy (olddateend, "----");
      }

use_display_window();
_clearscreen(_GWINDOW);
use_message_window();

set_title (actdata->title);
if ( nlines > 0)
      {
      DrawAxes (nlines, begin, end);
      if (Draw (nlines, begin, end)==-1)
          {
          _outtext ("Error during drawing process!!");
```

```
            }
        DrawLegend();
        if (new) AdvanceScale();
        DrawScale(lastscale);
        }
        else
        {
        display_status ("Unable to draw graph.");
        printf ("\a\a");
        wait_time(2);
        display_status (" ");
        use_message_window();
        return 0;
        }
return 1;

}


void InitializeGraphicalStructures (void)
{
spdata.initialisation = InitializeSpPlot;
strcpy (spdata.title, "Spirulina Compartment Graph");
strcpy (spdata.filename, "sp");
ntdata.initialisation = InitializeNtPlot;
strcpy (ntdata.title, "Nitrifying Compartment Graph");
strcpy (ntdata.filename, "nt");
simdata1.initialisation = InitializeSim1Plot;
strcpy (simdata1.title, "Simulation : Concentrations");
strcpy (simdata1.filename, "sim1");
simdata2.initialisation = InitializeSim2Plot;
strcpy (simdata2.title, "Simulation : BM mass fraction");
strcpy (simdata2.filename, "sim2");
simdata3.initialisation = InitializeSim3Plot;
strcpy (simdata3.title, "Simulation : Global formula");
strcpy (simdata3.filename, "sim3");
}

void SelectSpirulineGraph (void)
{
actdata = &spdata;
}

void SelectNitrifyingGraph (void)
{
actdata = &ntdata;
}

void SelectSimulation1Graph (void)
{
actdata = &simdata1;
}

void SelectSimulation2Graph (void)
{
actdata = &simdata2;
}

void SelectSimulation3Graph (void)
{
actdata = &simdata3;
}
```

```
void IncrementScale (void)
{
actdata->max[lastscale] = actdata->max[lastscale] * 2;
}

void DecrementScale (void)
{
actdata->max[lastscale] = actdata->max[lastscale] / 2;
}
```

Vars.c

```
#include "vars.h"

/*  #define _debugging_vars */

/*********************************************************************

      NAME              VARS.C

      AUTHOR            BINOIS C      (modified by FULGET N. ADERSA)

      DESCRIPTION
            access to data via gps functions
            and variables management

      UPDATES
            20-09-95

*********************************************************************/

static GPS_FILE *gps;
float selectcoefficient (char *name);
void filter(VARS *var);



/* ================================
      Initialisation of variables
==================================== */

void init_vars (void)
{
    init_vars_spirulina();
    init_vars_nitrogen();
    /* More reactor variables to be add here */
}

/*-------------------------------------
      acquisition of gps pointer
-------------------------------------*/
#ifndef ADERSA
void set_gps(GPS_FILE *gp)
#else
void set_gps(gp)
GPS_FILE *gp ;
#endif

      {
      gps=gp;
      }


/*-------------------------------------
      read function
-------------------------------------*/
#ifndef ADERSA
void read_var(VARS *read_var)
#else
void read_var(read_var)
VARS *read_var;
#endif
      {
```

```
      double  value;
#ifndef ADERSA
      double  read_gps(VARS *);
#else
      double  read_gps();
#endif
      int     file_rank;
      char    buffer[80];
#ifndef _debugging_vars        /* For trying independly of the plant
*/
      value=read_gps(read_var);
#else
      value = 1;
#endif
      if(value==-1)
            {
            file_rank=gps->rank;
            while(value==-1)
                  {
                  gps=activ_grp_gps();
                  value=read_gps(read_var);
                  if(file_rank==gps->rank)
                        {
                        sprintf(buffer,"Can not find %s in gps files
when reading",read_var->name);
                        display_error(buffer);
                        my_interrupt();
                        }
                  }
            }
      read_var->i++;
      read_var->i&=NB_SAMP;
      read_var->val[read_var->i]=value;
      read_var->value=value;
      }

/*---------------------------------------
      read gps sub_function
-------------------------------------*/

#ifndef ADERSA
double read_gps(VARS *read_vars)
#else
double read_gps(read_vars)
VARS *read_vars;
#endif
      {
      OGPS     read_ogps;
      IGPS     read_igps;
      int      read_count,j,k;
      int      ret_code,tag_found;
      char     buffer[9],name[9];
      double   read_value;

      read_value=0;
      for(read_count=0;read_count<1;read_count++)
      {
      switch(read_vars->tag_cmd)
            {
            case TAG:
                  {
                  ret_code=rd_gps(read_vars->name,&read_igps);
                  if(ret_code==-1)
                        {
```

```
                        error_gps();
                        return(-1);
                        }
                switch(ret_code)
                        {
                        case ISBIT:
                                {
                                if(read_igps.i.d.val_state==ACTIV)
                                        {read_value=1;}
                                else
                                        {read_value=0;}
                                if(!read_vars->update)
                                        {
                                        if(read_igps.i.d.act_state==ACTIV)
                                                {read_vars->max=1;}
                                        else
                                                {read_vars->max=0;}
                                        read_vars-
>dev_num=read_igps.h.dev_num;
                                        sprintf(read_vars->file,"%s",gps-
>file);
                                        read_vars->update=ON;
                                        }
                                break;
                                }
                        case ISABUS:
                                {
                                read_value=read_igps.i.bus.val;
                                if(!read_vars->update)
                                        {
                                        read_vars->min=read_igps.i.bus.l;
                                        read_vars->max=read_igps.i.bus.h;
                                        sprintf(read_vars-
>unit,"%s",read_igps.i.bus.unit);
                                        read_vars-
>dev_num=read_igps.h.dev_num;
                                        sprintf(read_vars->file,"%s",gps-
>file);
                                        read_vars->update=ON;
                                        }
                                break;
                                }
                        }
                break;
                }
        case CMD:
                {
                ret_code=set_cmd(read_vars->name,&read_ogps);
                if(ret_code==-1)
                        {
                        error_gps();
                        return(-1);
                        }
                switch(ret_code)
                        {
                        case OSBIT:
                        case OSREGIST:
                                {
                                sprintf(buffer,"%s is not a valid tag
name for MICON ..."
                                ,read_vars->name);
                                display_error(buffer);
                                break;
                                }
                        case OSMILOOP:
                              {
                              read_value=read_ogps.o.ml.val;
                              read_vars->out=read_ogps.o.ml.out;
                              read_vars->sp=read_ogps.o.ml.sp;
                              if(!read_vars->update)
                                    {
                                    read_vars-
>min=read_ogps.o.ml.spmin;
                                    read_vars-
>max=read_ogps.o.ml.spmax;
                                    sprintf(read_vars-
>unit,"%s",read_ogps.o.ml.spunit);
                                    read_vars-
>dev_num=read_ogps.h.dev_num;
                                    sprintf(read_vars->file,"%s",gps-
>file);
                                    read_vars->update=ON;
                                    }
                              break;
                              }
                        case OSMILOC:
                              {
                              read_value=read_ogps.o.loc.val;
                              read_vars->sp=read_ogps.o.loc.val;
                              if(!read_vars->update)
                                    {
                                    read_vars-
>min=read_ogps.o.loc.locmin;
                                    read_vars-
>max=read_ogps.o.loc.locmax;
                                    sprintf(read_vars-
>unit,"%s",read_ogps.o.loc.locunit);
                                    read_vars-
>dev_num=read_ogps.h.dev_num;
                                    sprintf(read_vars->file,"%s",gps-
>file);
                                    read_vars->update=ON;
                                    }
                              break;
                              }
                        case OSMIVD:
                              {
                              read_value=read_ogps.o.vd.val;
                              read_vars->sp=read_ogps.o.vd.val;
                              if(!read_vars->update)
                                    {
                                    read_vars-
>dev_num=read_ogps.h.dev_num;
                                    sprintf(read_vars->file,"%s",gps-
>file);
                                    read_vars->update=ON;
                                    }
                              break;
                              }
                        }
                  break;
                  }
            default:
                  {
                  tag_found=OFF;
                  j=nbtags+nbcommands;
                  for(k=0;k<j;k++)
                        {
```

```
                        sprintf(buffer,"%s",gettags(k));
                        if(strcmp(buffer,read_vars->name)==0)
                                {
                                tag_found=ON;
                                if(k<nbtags)
                                        {
                                        ret_code=rd_gps(read_vars-
>name,&read_igps);
                                        read_vars->tag_cmd=TAG;
                                        }
                                else
                                        {
                                        ret_code=set_cmd(read_vars-
>name,&read_ogps);
                                        read_vars->tag_cmd=CMD;
                                        }
                                if(ret_code==-1)
                                        {
                                        error_gps();
                                        return(-1);
                                        }
                                else
                                        {
                                        read_vars->type=ret_code;
                                        read_count--;
                                        }
                                break;
                                }
                        }
                if(!tag_found)
                        {
                        return(-1);
                        }
                }
        }
    }
    if(read_value==-1)
        {
        return(-0.9999999);
        }
    return(read_value);
    }

/*---------------------------------------
    write function
-------------------------------------*/


#ifndef ADERSA
void write_var(VARS *write_var)
#else
write_var(write_var)
VARS *write_var;
#endif

    {
    int     file_rank,ret_code;
    char    buffer[80];

#ifndef __Monitoring        /* It gives us the chance to have
monitoring stations */

#ifndef _debugging_vars
    ret_code=write_gps(write_var);
```

```
#else
      ret_code = 0;
#endif
      if(ret_code==-1)
            {
            file_rank=gps->rank;
            while(ret_code==-1)
                  {
                  gps=activ_grp_gps();
                  ret_code=write_gps(write_var);
                  if(file_rank==gps->rank)
                        {
                        sprintf(buffer,"Can not find %s in gps files
when writing",write_var->name);
                        display_error(buffer);
                        my_interrupt();
                        break;
                        }
                  }
            }
#endif
      }



/*---------------------------------------
      write gps sub_function
--------------------------------------*/

#ifndef ADERSA
int write_gps(VARS *write_vars)
#else
int write_gps(write_vars)
VARS *write_vars;
#endif
      {
      OGPS    write_ogps;
      S_USER  write_s_user;
      int     ret_code,tag_written;
      time_t  ltime;
    char    buffer[80];

      tag_written=OFF;
      while(!tag_written)
      {
      switch(write_vars->tag_cmd)
            {
            case TAG:
                  {
                  sprintf(buffer,"Can't write the TAG %s
...",write_vars->name);
                  display_error(buffer);
                  my_interrupt();
                  break;
                  }
            case CMD:
                  {
                  ret_code=set_cmd(write_vars->name,&write_ogps);
                  if(ret_code==-1)
                        {
                        error_gps();
            return(-1);
                        }
                  switch(ret_code)
```

```
                              {
                              case OSBIT:
                              case OSREGIST:
                                    {
                                    sprintf(buffer,"%s is not a valid tag
name for MICON ..."
                                    ,write_vars->name);
                                    display_error(buffer);
                                    break;
                                    }
                              case OSMILOOP:
                                    {
                                    if((write_ogps.o.ml.state&16)==16)
                                          {
                                          write_s_user.mic.mode=MICLOCREM;
                                          write_s_user.mic.status_sta=ACTIV;

      if(wr_gps(&write_ogps,&write_s_user)==-1)
                                                {
                                                error_gps();
                                                }
                                          break;
                                          }
                                    if((write_ogps.o.ml.state&1)==0)
                                          {
                                          write_s_user.mic.mode=MICAUTO;
                                          write_s_user.mic.status_sta=ACTIV;

      if(wr_gps(&write_ogps,&write_s_user)==-1)
                                                {
                                                error_gps();
                                                }
                                          break;
                                          }
                                    write_s_user.mic.status_sta=INACTIV;
                                    write_s_user.mic.status_out=INACTIV;
                                    write_s_user.mic.status_ra=INACTIV;
                                    write_s_user.mic.status_bi=INACTIV;
                                    write_s_user.mic.status_loc=INACTIV;
                                    write_s_user.mic.status_vd=INACTIV;
                                    write_s_user.mic.val_sp=write_vars->sp;
                                    write_s_user.mic.status_sp=ACTIV;
                                    tag_written=ON;
                                    break;
                                    }
                              case OSMILOC:
                                    {
                                    write_s_user.mic.val_loc=write_vars->sp;
                                    write_s_user.mic.status_loc=ACTIV;
                                    tag_written=ON;
                                    break;
                                    }
                              case OSMIVD:
                                    {
                                    write_s_user.mic.val_vd=(int)write_vars-
>sp;
                                    write_s_user.mic.status_vd=ACTIV;
                                    tag_written=ON;
                                    break;
                                    }
                              }
                  break;
                  }
            default:
```

```
                        {
                        read_var(write_vars);
                        }
                }
        }
        if(wr_gps(&write_ogps,&write_s_user)==-1)
                {
                error_gps();
                time(&ltime);
                sprintf(buffer,"When writing %s at time %s",write_vars-
>name,
                ctime(&ltime));
                display_error(buffer);

        return(0);
                }
        else
                {
        return(0);
                }
    }

/*-------------------------------------
        errors management
-------------------------------------*/

error_gps()
        {
        switch (gpserror)
                {
                case ERDOS:
                        {
                        display_error("A DOS problem has occured ...\n");
                        my_interrupt();
                        break;
                        }
                case ENETW:
                        {
                        display_error("Network or Mailbox fault ...\n");
                        break;
                        }
                case INVALID_FGPS:
                        {
                        display_error("Incorrect GPS file ...\n");
                        my_interrupt();
                        break;
                        }
                case ETAGCMD:
                        {
                        break;
                        }
                case ETAGTYP:
                case ECMDTYP:
                        {
                        display_error("Unknown CMD or TAG ...");
                        my_interrupt();
                        break;
                        }
                case ECONV:
                        {
                        display_error("Floating point conversion
error:ignored ...");
                     /* my_interrupt();*/
                        break;
                }
          case EEQUIP:
                {
                display_error("Unknown PLC protocol ...");
                my_interrupt();
                break;
                }
          case ENUMPLC:
                {
                display_error("Invalid device number ...");
                my_interrupt();
                break;
                }
          default:
                {
                display_error("Unidentified error ...");
                my_interrupt();
                break;
                }
          }
    }

/*---------------------------------------
      check if mailbox is refresh
--------------------------------------*/

check_network()
      {
      int counter=0;  /* If the net is not working ... we generate a
                      infinit loop */
    char buffer[300];
      int rt;


      display_status("Checking Network ...");
      while((rt=garde(101,1)) && (counter++ < MAX_CHECK_NETWORK))
            {
/*          sprintf (buffer, " garde(..) returned -1, error
(%i)",counter);
      switch(gpserror) {
                  case  ENETW : strcat (buffer, "ENETW "); break;
                  case  ERDOS : strcat (buffer, "ERDOS "); break;
                  case  ENUMPLC : strcat (buffer, "ENUMPLC "); break;
                  }
       if (((counter+1) % 20) == 0)
            {
            strcat (buffer, "\n\t\t revising network recomended if it
happens so often");
            log_file (buffer);
            }*/
      }
    if (counter >= MAX_CHECK_NETWORK) log_file ("Network test not
passed!");
      display_status(" ");
      }


/* =============================
    Filter functions ...
=================================*/

/* ====================================
    Main filter function
==================================*/
```

```
void filter(VARS *var)
{
float alpha;
float realval;

alpha = selectcoefficient (var->name);

realval = var->val[var->i];      /* Mesured value */
var->val[var->i] = alpha * var->val[ var->i - 1 % (NB_SAMP + 1)] +
       (1-alpha) * realval;
var->value = var->val[var->i];

}


/*==================================
    Selecting coefficients
==================================*/

float selectcoefficient (char *name)
{
if (!strcmp (name, "LOOP0107")) return ALPHAFILTERcxa;
if (!strcmp (name, "LOOP0103")) return ALPHAFILTERnitrate;
if (!strcmp (name, "LOOP0105")) return ALPHAFILTEREb;
if (!strcmp (name, "LLOP0104")) return ALPHAFILTERpH;

return 1;
}
```

Qbs.c

```
      /*
      qbs.c

      Main programme for the estimation of the quality of the biomass
of
      the photoautotrophic (spirulina) compartment

      J.J. Leclercq
      ADERSA
      April 1997

      */

#include "parmodel.h"
#include "pargene.h"
#include "math.h"
#include <stdio.h>
extern void modspiru();
extern void modspiru2();

#define FCr0 "cr0.txt"
#define FFRH "frh.txt"
#define FdilH "dilh.txt"
#define FCiH "cih.txt"
#define FCrH "crh.txt"
#define FmasbioH "fmasbioh.txt"
#define FchonspH "chonspH.txt"

void PrintRes (double H);


main()
{
      double H, tsim;
      short i, i1, i2, choisim;
    FILE *pf, *fCr0, *fFRH, *fdilH, *fCiH;
    double Cr0[nsig+1];

/*    Choice of the example of simulation */
      choisim = 1;

/*
In this part of the main programme, the user has to define
      1_ the duration of the simulation : tsim (expressed in hours)
      2_ the initial concentrations in the reactor (in kg/m3)
      3_ the time variations of the inputs along the horizon of
simulation H :
            FRH : incident radient energy flux (W/m2)
            dilH : dilution rate (1/h)
            concentration (kg/m3) in the incoming flow of :
            CiH(1) : total biomass
            CiH(2) : active biomass
            CiH(3) : chorophyll
            CiH(4) : phycocyanins
            CiH(5) : proteins
            CiH(6) : nitrate
            CiH(7) : sulfate
            CiH(8) : vegetative biomass
            CiH(9) : exopolysaccharide

*/
   if (choisim == 1)
   {
      /* ======================================================
         Simulation according to TN19.3 , Appendix 7 , Figure 8
         ====================================================== */

      /* 1_ initialization of the duration of the simulation */
    tsim = 500.;          /* duration of the simulation (in hours) */
      H = tsim / dt;          /* length of simulation (in sampling
periods)
                                dt : sampling period defined in pargene.h */
      if(nT < H)  /* dimension test */
        {
        printf("******* The simulation duration is too long\n");
        printf("versus the dimension nT. Increase nT in pargene.h
*******\n");
        exit(1);
        }

      /* 2_ initial concentrations in the reactor */
    Cr0[2] = .1;            /* active biomass */
    Cr0[9] = .02;           /* exopolysaccharide */
    Cr0[6] = .8;            /* nitrate */
    Cr0[7] = .2;            /* sulfate */
    Cr0[8] = Cr0[2];        /* vegetative biomass */
    Cr0[1] = Cr0[2] + Cr0[9];   /* total biomass */
    Cr0[3] = .01 * Cr0[2];      /* chlorophyll */
      Cr0[4] = .162 * Cr0[2];       /* phycocyanin */
      Cr0[5] = .684 * Cr0[2];       /* protein */


    /* Creating the test file Cr0 */
    fCr0 = fopen (FCr0, "w");
    if (fCr0 == NULL) {
        printf ("\nNo puc obrir el fitxer Cr0\n");
        exit (-1);
        }
    for (i =1; i <= 9; i++) {
        fprintf (fCr0, " %lf\n ", Cr0[i]);
        }
    fclose (fCr0);

      /* 3_ time variation of the inputs */
    i1 =(int)(50. / dt);    /* dilution rate step at time = 50 h */
      i2 = (int)(250. / dt);  /* incident flux step at time = 250 h */

    /* Create FRH file */
    fFRH = fopen (FFRH, "w");
    for (i=0; i<=i2; i++) {
        /*       FRH[i] = 50.; */
        fprintf (fFRH, "50.\n");
        }
    for (i=i2+1; i<=H; i++) {
        /*       FRH[i] = 100.;      /* incident flux step */
        fprintf (fFRH, "100.\n");
        }
    fclose (fFRH);

    fdilH = fopen (FdilH, "w");
    for (i=0; i<=i1; i++) {
        /*       dilH[i] = 0.;     */
        fprintf (fdilH, "0.\n ");
        }
    for (i=i1+1; i<=H; i++) {
        /*        dilH[i] = 0.05;      /* dilution rate step */
        fprintf (fdilH, "0.05\n ");
        }
    fclose (fdilH);

    /* Create CiH file */
    fCiH = fopen (FCiH, "w");
      for (i=0; i<=H; i++)
      {
            /* constant concentration (kg/m3) in the incoming flow of :
*/
    /*        CiH[i][1] = .0;       /* total biomass */
    /*        CiH[i][2] = .0;       /* active biomass */
    /*        CiH[i][3] = .0;       /* chorophyll */
    /*        CiH[i][4] = .0;       /* phycocyanins */
    /*        CiH[i][5] = .0;       /* proteins */
    /*        CiH[i][6] = Cr0[6]; /* nitrate */
    /*        CiH[i][7] = Cr0[7]; /* sulfate */
    /*        CiH[i][8] = .0;       /* vegetative biomass */
    /*        CiH[i][9] = .0;       /* exopolysaccharide */
        fprintf (fCiH, ".0 .0 .0 .0 .0 %lf %lf .0 .0\n", Cr0[6],
Cr0[7]);
      }
    fclose (fCiH);
   }
   else if (choisim == 2)
   {
      /* ===================================================
         Simulation of a batch (dilution rate is null)
         ===================================================  */

      /* 1_ initialization of the duration of the simulation */
      tsim = 50.;         /* duration of the simulation (in hours) */
      H = tsim / dt;           /* length of simulation (in sampling
periods)
                                 dt : sampling period defined in pargene.h */
      if(nT < H)  /* dimension test */
        {
        printf("******* The simulation duration is too long\n");
        printf("versus the dimension nT. Increase nT in pargene.h
*******\n");
        exit(1);
        }

      /* 2_ initial concentrations in the reactor */
      Cr0[2] = .1;                  /* active biomass */
      Cr0[9] = .02;                 /* exopolysaccharide */
      Cr0[6] = .8;                  /* nitrate */
      Cr0[7] = .2;                  /* sulfate */
      Cr0[8] = Cr0[2];        /* vegetative biomass */
      Cr0[1] = Cr0[2] + Cr0[9];     /* total biomass */
      Cr0[3] = .01 * Cr0[2];        /* chlorophyll */
      Cr0[4] = .162 * Cr0[2];       /* phycocyanin */
      Cr0[5] = .684 * Cr0[2];       /* protein */

    fCr0 = fopen (FCr0, "w");
    if (fCr0 == NULL) {
        printf ("\nNo puc obrir el fitxer Cr0\n");
        exit (-1);
        }
    for (i =1; i <= 9; i++) {
```

```
    fprintf (fCr0, " %lf\n", Cr0[i]);
    }
  fclose (fCr0);

   /* 3_ time variation of the inputs */
  for (i=0; i<=H; i++) {
    /*        FRH[i] = 50.; */
    fprintf (fFRH, "50.\n");
    }
  for (i=0; i<=H; i++) {
    /*        FRH[i] = 50.; */
    fprintf (fFRH, "50.\n");
    }
   for (i=0; i<=H; i++)
   {
      /* constant concentration (kg/m3) in the incoming flow of :
*/
  /*    CiH[i][1] = .0;      /* total biomass */
  /*    CiH[i][2] = .0;      /* active biomass */
  /*    CiH[i][3] = .0;      /* chorophyll */
  /*    CiH[i][4] = .0;      /* phycocyanins */
  /*    CiH[i][5] = .0;      /* proteins */
  /*    CiH[i][6] = Cr0[6]; /* nitrate */
  /*    CiH[i][7] = Cr0[7]; /* sulfate */
  /*    CiH[i][8] = .0;      /* vegetative biomass */
  /*    CiH[i][9] = .0;      /* exopolysaccharide */
   }
  fprintf (fCiH, ".0 .0 .0 .0 .0 %lf %lf .0 .0\n", Cr0[6], Cr0[7]);
 }

/*
End of the initialization done by the user
*/

   if(nZ < nstep)    /* dimension test */
    {
    printf("******* The number of integration steps is too
big\n");
    printf("versus the dimension nZ. Increase nZ in pargene.h
*******\n");
    exit(1);
    }

   /*
   Computation on the time horizon H of :
   . concentrations of the compounds in the reactor : 'CrH'
   . mass fraction of the biomass : fmasbioH
   . global formula (CHONSP) of the biomass : chonspH
   */
  modspiru2 (H, FCr0, FFRH, FdilH, FCiH, FCrH, FmasbioH, FchonspH);
  PrintRes(H);

}


void PrintRes (double H)
{
int i;
FILE *pf;
FILE *fCrH, *fFRH, *fdilH, *fmasbioH, *fchonspH;
double v1, v2, v3, v4, v5, v6, v7, v8, v9;
double frh, dilh;
  /*
  Saving results into 3 files *.res :
    1_ concentrations of the compounds in the reactor : conc.res
    2_ mass fraction of the biomass : compo.res
    3_ global formula (CHONSP) of the biomass : glob.res
    */


    fCrH = fopen (FCrH, "r");
    fFRH = fopen (FFRH, "r");
    fdilH = fopen (FdilH, "r");

    pf = fopen("conc.res", "w");
    for (i=0; i<=H; i++)
    {
    fscanf (fFRH, " %lf", &frh);
    fscanf (fdilH, " %lf", &dilh);
    fscanf (fCrH, "%lf %lf %lf %lf %lf %lf %lf %lf %lf", &v1, &v2,
&v3,
            &v4, &v5, &v6, &v7, &v8, &v9);
    fprintf(pf,"%10.2f %12.5e %12.5e %12.5e %12.5e %12.5e %12.5e
%12.5e %12.5e %12.5e %12.5e %12.5e\n",
        i*dt,frh,dilh, v1, v2, v3, v4, v5, v6, v7, v8, v9);
    }
    fclose(pf);
    fclose (fCrH);
    fclose (fFRH);
    fclose (fdilH);

    fmasbioH = fopen (FmasbioH, "r");
    pf = fopen("compo.res", "w");
    for (i=0; i<=H; i++)
    {
    fscanf (fmasbioH, "%lf %lf %lf %lf %lf %lf", &v1, &v2, &v3,
            &v4, &v5, &v6);
    fprintf(pf,"%10.2f %12.5e %12.5e %12.5e %12.5e %12.5e
%12.5e\n",i*dt,
        v1,v2,v3,v4, v5, v6);
    }
    fclose(pf);
    fclose (fmasbioH);

    fchonspH = fopen (FchonspH, "r");

    pf = fopen("glob.res", "w");
    for (i=0; i<=H; i++)
    {
    fscanf (fchonspH, "%lf %lf %lf %lf %lf", &v1, &v2, &v3,
            &v4, &v5);
    fprintf(pf,"%10.2f %12.5e %12.5e %12.5e %12.5e %12.5e\n",i*dt,
        v1, v2, v3, v4, v5);
    }
    fclose(pf);
    fclose (fchonspH);
}
```